

Nº10 – Noviembre 2004

zona PC

CREACIÓN DE FICHEROS TZX EN GNU/LINUX

analisis

INTO THE EAGLES NEST
CAR CHASE
REVEAL

al descubierto

MISTERIO EN LA CATEDRAL

EL JUEGO DE LA VIDA DE JOHN CONWAY

...y mucho más

www.speccy.org/magazinezx



Año II Nº 10 Noviembre 2004





Redacción:

Santiago Romero (SROMERO).
Federico Álvarez (FALVAREZ).
Pablo Suau (SIEW).
Miguel A. García Prada (DEVIL_NET).

Ilustración de Portada:

Juanje Gómez (DEV).

Colaboraciones en este número:

Josetxu Malanda (HORACE).
S.T.A.R.

Maquetación en PDF

Álvaro Alea (ALEASOFT)

Contacto:

magazine@speccy.org



Redacción de MAGAZINE ZX

Coincidiendo con la llegada del frío, os ofrecemos una nueva entrega de MagazineZX, la décima desde que comenzamos nuestra andadura en el mundo editorial online, cubriendo el pequeño hueco que había en cuanto a publicaciones relativas al Spectrum.

En este décimo número os ofrecemos de nuevo material apto para todos los aficionados del Spectrum, desde los jugones hasta los programadores.

En cuanto al apartado lúdico, los análisis del conocidísimo Into The Eagles Nest, así como de los no tan conocidos Car Chase y Reveal. De la mano de Miguel, dejamos al descubierto "Los Extraordinarios Casos del Dr. Van Halen: Misterio en la Catedral".

Quienes quieran ayudar a proyectos como SPA2, El Trastero o WorldOfSpectrum desde sus máquinas con Sistema Operativo Linux encontrarán especialmente interesante el artículo de ZonaPC sobre "Creación de ficheros de cinta TZX en GNU/Linux". De la mano de Horace descubriremos todos los pasos necesarios para volver a fichero TZX esas cintas que todavía no están "preservadas" en ficheros digitales.

S.T.A.R. vuelve a sorprendernos de nuevo con su columna de opinión, que no dejará a nadie indiferente. Este mes: el lenguaje, la manera de decir las cosas, la manipulación de la información, y cómo expresar (y leer) opiniones que difieren de las generales en los medios escritos como MagazineZX o las news.

La cuota mensual de programación de nuestra revista viene marcada por 2 artículos basados en ejemplos prácticos comentados. Por un lado, un juego completo en BASIC (ZXSNAKE) comentado y desgranado de forma que los lectores que comienzan a programar vean de qué manera se implementa en el BASIC del Spectrum el clásico juego de la serpiente.

Finalmente, para aquellos que comienzan a programar en C en nuestro Spectrum nada mejor que el ejemplo completo comentado en nuestra sección de Z88DK: El Juego de la Vida de John Conway. Un automata celular basado en sencillas reglas que causó estragos entre los computadores y microcomputadores de la década de los 70. Sromero nos mostrará las reglas del "Juego de la Vida", el pseudocódigo que define el autómata celular, y una implementación en C para nuestro Spectrum del famoso algoritmo (incluyendo un fichero TAP para que los lectores puedan probarlo en su Spectrum real o en un emulador).

Como puede verse, otra edición cargados de contenidos por y para nuestros lectores.✍



Redacción de MAGAZINE ZX

Desde el anterior número de MagazineZX han aparecido muchas y jugosas novedades en el mundo Spectrum, empezando por nuevas publicaciones (sí, en plural), pasando por cambios en emuladores tan importantes como RealSpectrum, rumores de nuevos juegos en desarrollo por el WSS Team y acabando en una nueva edición del Concurso de Programación en BASIC 2004. A continuación os contamos las noticias más relevantes:

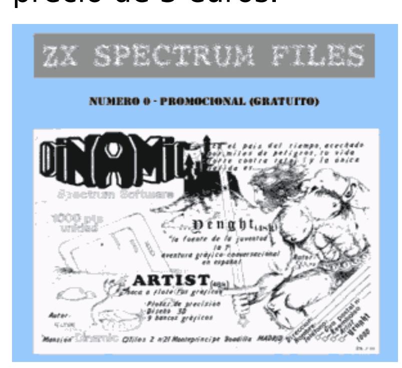
First Generation: nueva revista sobre videojuegos

MATRANET nos presenta la nueva revista First Generation. Esta nueva revista en papel, de pago, y que podéis obtener directamente en la web de Matranet cubre diferentes sectores del panorama de los videojuegos y sistemas Vintage, y en su primera entrega incluye un póster central de nuestro querido Spectrum 48K "gomas". La revista será bimestral, y el número 1 cuenta con 32 páginas a todo color por el módico precio de 4 euros. Esperamos que el Spectrum tenga mucha relevancia dentro de la revista y que podamos encontrar en todos los números contenidos relacionados con nuestra pequeña máquina para animarnos a comprarla.

Nuevo fanzine ZX Spectrum Files

Continúa ampliándose el espectro informativo del mundo del Spectrum (valga la redundancia) gracias a la aparición de un nuevo fanzine en formato PDF: ZX Spectrum Files. Este fanzine, totalmente en Español, viene de la mano de Ignacio Prini García, que muchos de nosotros todavía recordamos por sus colaboraciones en la revista Microhobby.

El fanzine se distribuye en formato PDF junto a utilidades extra y material relacionado con los comentarios de la revista en CDROM, que recibiréis en vuestro domicilio al precio de 3 euros.



Portada del número 0 de ZXSF

Alojado en Microhobby.com tenéis disponible el número 0 del Fanzine para poder evaluarlo y para permitir decidiros si queréis o no realizar una suscripción anual.

Este primer número ha sido realizado integramente por Ignacio Prini, pero suponemos que irá ganando en calidad y diversidad conforme se vaya apuntando más gente a colaborar en él.

Seguimiento del Speccy Tour 2004

Desde hace varias semanas se puede seguir en tiempo real la clasificación del Speccy Tour 2004 desde la página principal del portal de Spectrum Speccy.org. En el momento de redacción de este artículo, los participantes que copan los 5 primeros puestos son Néstor Lucas Martínez (Humitsec), Santiago Eximeno, Ivan Ruiz Etxabe, Eduardo Yáñez Parareda y Malc74. Los 38 participantes se están empleando a fondo, pero

estos cinco están siendo de momento los "abonados" a los primeros puestos. Los siguientes españoles mejor posicionados son David Muriel, Jaime Tejedor Gómez, Pablo Suau y Federico J. Álvarez Valero, en las posiciones 11, 12, 15 y 17 respectivamente. Como véis, la participación española este año es masiva y la calidad de los participantes les hace posicionarse a casi todos ellos en la parte alta de la tabla de clasificación.

Podéis seguir la clasificación completa del Speccy Tour en la sección de Clasificaciones de su Web.

bcnparty'100: Reunión de Usuarios en Barcelona

El fin de semana largo (gracias al puente festivo) del 29 de Octubre al 1 de Noviembre se celebró en Barcelona la bcnparty '100, una reunión de usuarios de diferentes sistemas en formato "Party creativa", con competiciones de todo tipo, programación tanto en sistemas modernos como vintage, incluyendo el ya famoso "lanzamiento de diskette", todo por un precio entre 25 y 35 euros.

Más información en su página web.

Bug Fixing del emulador RealSpectrum

El día 2 de Noviembre apareció una nueva versión del emulador Realspectrum de RAMSOFT que corrige bugs existentes en versiones anteriores. El sobrenombre "Finale" de esta nueva release 14B probablemente indique que RAMSOFT no continuará con el desarrollo activo de las versiones para MSDOS y se centrará en las versiones Windows y una posible nueva versión para Linux. Mientras tanto podemos disfrutar de la última versión del emulador en su web.

Rumores de nuevo juego de WSS Team

Y más sobre programación: Nuestro compañero MATRA nos ha dejado a todos intrigados con los rumores sobre un posible nuevo juego para Spectrum programado por WSS Team (los autores de Flash Beer Trilogy y TV Game) para finales del 2004 (o principios del 2005). No sólo eso, sino que además nos pone los dientes largos haciéndonos saber que tiene conocimiento de posibles futuros proyectos del WSS Team que resultan muy interesantes a su modo de ver.

Estaremos atentos a la web de WSS Team y a la posible distribución en España por parte de MATRA de cualquier novedad.



Concurso de Programación de juegos en BASIC 2004

Radastan, habitual de es.comp.sistemas.sinclair, vuelve a organizar una nueva entrega de su Concurso de Programación de Juegos en Basic. Este año se aceptarán programas tanto para Spectrum como para plataformas MSX1, Amstrad CPC 464 y 6128, tanto en cinta como en disco, entrando a participar en 2 categorías diferentes. Por un lado competirán aquellos programas que utilizan los recursos no gráficos predefinidos de los cuales provea el ordenador, sin utilizar pantallas de carga ni gráficos similares (aunque se permiten llamadas a la ROM para acelerar la ejecución del juego), mientras que por otro lado se podrán utilizar gráficos definidos por el usuario (UDG) y una pantalla de carga como fondo del programa o juego. Se hacen necesarias estas 2 categorías ya que resultaría muy complicado que programas del primer grupo se impusieran gráficamente sobre los del segundo.

Cabe recordar que, en todo caso, los programas presentados al concurso deben ceñirse al BASIC de la máquina, sin utilizar rutinas externas en ensamblador ni recursos extra de la arquitectura destino. Además, deben de estar programados exclusivamente para el concurso, con lo cual no se podrán presentar programas rescatados de otros concursos o realizados anteriormente.

El concurso finalizará el día 23 de Diciembre a las 24:00 y los premios consisten en suscripciones a diferentes publicaciones relacionadas con los videojuegos, o incluso consolas clónicas (tipo NES o Tetris). Más información en la web del concurso.

Emulador ASpectrum portado a GP32

Otra noticia relacionada con la emulación (esta vez portátil) nos va a permitir llevarnos el Spectrum a cualquier parte. Los afortunados poseedores de una consola GP32 podrán instalar en ella la versión GP32 de Aspectrum, el emulador desarrollado por Santiago Romero y Álvaro Alea (entre otros). Nuestro compañero Rlyeh no sólo ha portado el emulador a la portátil coreana, sino que además le ha hecho mejoras que se han fundido con las realizadas por Álvaro Alea y que han convergido de forma que Aspectrum tiene ahora soporte de 128K, una mejor emulación de la representación de los gráficos y algunos bugs corregidos.

Nueva actualización de SPA2

Y para acabar, una noticia relacionada con una de las webs más importantes del panorama Español: The Spectrum Spanish Archive ha actualizado contenidos después de 9 meses. El cambio más importante que presenta SPA2 es que ahora también se hospedarán ficheros Z80 y TAP (no sólo ficheros TZX) y no sólo almacenará programas comerciales, sino también programas caseros o de dominio público españoles.



En esta ocasión la actualización ha sido muy jugosa, con 22 nuevos ficheros Z80, 146 nuevos TAP, 168 nuevos TZX, 1 nuevo DSK, 331 nuevas carátulas y 175 nuevas instrucciones de juegos.

Podéis descargar todas las novedades a fecha 23 de Octubre desde la web de SPA2.✍

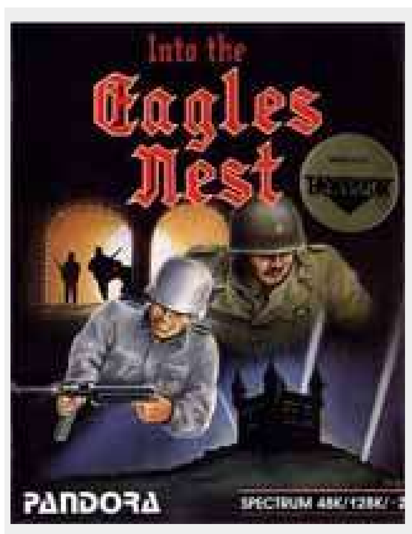
LINKS

- Matranet:
  http://www.matranet.net/
- ZX Spectrum Files:
  http://www.microhobby.com/zxsf/pagina_1.htm
- Clasificación en tiempo real del ST04:
  http://www.speccy.org/st04/scoreboard.php
- bcnparty'100:
  http://www.bcnparty.org/
- Real Spectrum:
  http://www.ramsoft.bbk.org/realspec.html
- WSS Team:
  http://wss.sinclair.hu/
- Concurso de BASIC 2004:
  http://www.redeya.com/bytemaniacos/concurso2004/index.html
- SPA2:
  http://www.speccy.org/spa2/spanish/whatsnew.htm



En este número presentamos el análisis de Into The Eagles Nest, a cargo de SROMERO (NoP), y de dos títulos menos conocidos, Car Chase y Reveal, comentados por FALVAREZ. Esperamos haber conseguido descubriros algún título desconocido para vosotros y que los que hemos escogido sean de vuestro agrado.

INTO THE EAGLES NEST



Título	Into The Eagles Nest
Género	Acción / Aventura
Año	1986
Máquina	48-128K
Jugadores	1 Jugador
Compañía	Pandora
Autor	Kevin Parker y Robin Chapman
Otros comentarios	

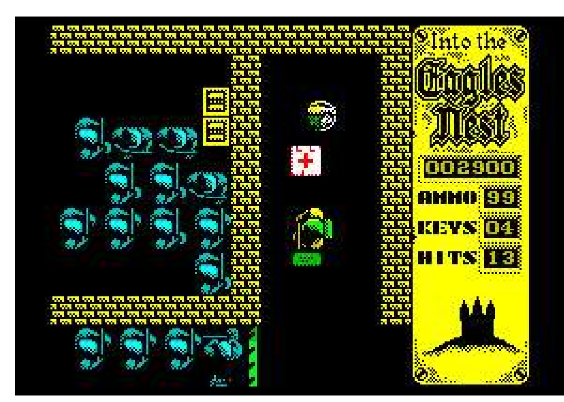
- Your Sinclair http://www.ysrnry.co.uk/articles/intotheeaglesnest.htm
- Crash ftp://ftp.worldofspectrum.org/pub/sinclair/magazines/Crash/Issue39/Pages/Crash3900109.jpg
- Microhobby ftp://ftp.worldofspectrum.org/pub/sinclair/magazines/MicroHobby/Issue156/Pages/MicroHobby15600022.jpg

Estamos en plena Segunda Guerra Mundial y somos uno de los mejores agentes aliados. Nuestra misión es infiltrarnos en una fortaleza casi inexpugnable (el Nido del Águila) y sabotearla, ya que es uno de los principales centros de refuerzo de tropas y munición del Ejército Alemán.



Nuestro héroe en el punto de partida

No somos los primeros que lo intentamos; 3 agentes aliados fueron capturados antes que nosotros intentando realizar la misma misión. Aparte de la misión principal de volar todas y cada una de las plantas del castillo, nuestro objetivo secundario será rescatar sanos y salvos a los prisioneros. El último de los comandos capturados fue capaz de dejar todos los explosivos colocados en los lugares apropiados, pero fue detectado y reducido antes de poder detonarlos.



La comida y el botiquín

El juego es bastante largo y consta de 4 misiones diferentes, que podrían considerarse casi como jugar a 4 variantes del juego usando el mismo motor:

- Misión 1: Detonar las 8 plantas del castillo (6 pisos, planta baja y sótano), planta a planta, recogiendo y utilizando los explosivos y los detonadores que encontraremos en cada una de las plantas, comenzando desde la parte superior del mismo.

- Misión 2: Rescatar a un prisionero de la planta sexta y llevarlo hasta el punto inicial.
- Misión 3: Rescatar a un prisionero, pero puede estar en cualquier planta (hay que buscarlo), y llevarlo al punto inicial del juego.
- Misión 4: Detonar las 8 plantas del castillo, pero con menos recursos disponibles para el jugador (menos botiquines y comida). Ni qué decir tiene que el juego será mucho más difícil.

Así pues, en la primera misión nos centramos exclusivamente en la destrucción del castillo mientras que para rescatar a los cautivos jugaremos el resto de las misiones, ascendiendo plantas hasta que encontremos uno (lo reconoceremos por utilizar el mismo gráfico que el jugador, pero sin armas). El rehén se considerará rescatado cuando lo llevemos a la planta baja (Ground Floor).



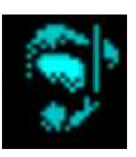
Aspecto del menú principal del juego

Además, estas 4 misiones se combinan con 2 posibles niveles de dificultad seleccionables en el menú principal: EASY y HARD (fácil y difícil). En el primer nivel, con cada disparo podremos abatir un soldado nazi, mientras que en el nivel difícil serán necesarios 2 disparos para hacerlo. Obviamente, nuestras reservas de munición se reducirán mucho más rápidamente en el nivel difícil.

Elementos del juego

Veamos los diferentes elementos que aparecen en el juego:

- Los enemigos: Los soldados nazis se cuenta por cientos en este juego, y se abalanzarán sobre nosotros para quitarnos energía disparándonos o golpeándonos.



- Impactos (hits): Nuestro cuerpo es capaz de aguantar 50 impactos de los enemigos, ya sean disparos o choques directos contra los soldados nazis. Deberemos vigilar este marcador para evitar nuestra muerte ya que sólo contamos con una vida.



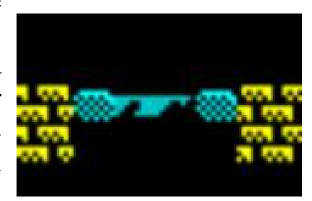
- Munición: Empezamos el juego con 99 municiones, pero podemos recargar nuestro arma gracias a los cartuchos de munición que encontraremos en el juego. Cada cartucho nos proporcionará 15 disparos extra.

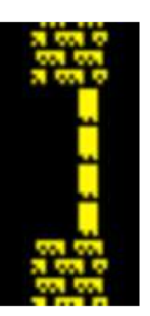


- Llaves: Diseminadas por el mapeado, las llaves nos permiten abrir cierto tipo de puertas.

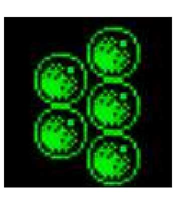


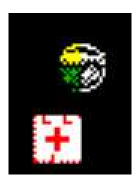
- Puertas sin cerrar: Algunas puertas pueden ser abiertas simplemente disparando sobre ellas. Conviene hacerlo a distancia para evitar contactos con soldados alemanes que estén al otro lado de las mismas.



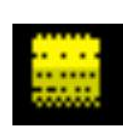


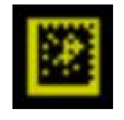
- Puertas cerradas: El resto de puertas del juego requerirá de una llave para poder ser abiertas. Es importante que pensemos antes de abrir cualquier puerta y que abramos aquellas puertas que nos permitan recoger más llaves, ya que si nos quedamos sin llaves corremos el riesgo de no poder continuar.
- Barriles: Los barriles son objetos que podemos destruir con disparos (de hecho, en algunas ocasiones tendremos que hacerlo para poder avanzar), y que sirven también para protegernos de los enemigos.





- Comida y Botiquines: nos hacen recuperar energía, reduciendo nuestros hits. El botiquín es mucho más efectivo, pues reduce a cero el marcador de hits.
- Cajas cerradas: Las cajas cerradas se abren al dispararles, y pueden contener otros objetos. Pero cuidado, no sólo hay objetos buenos, también pueden contener explosivos que nos hagan perder la vida.



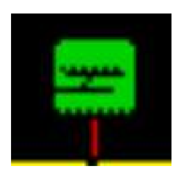


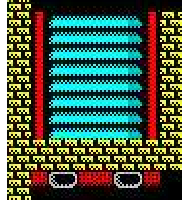
- Puntos extra: Existen diferentes objetos en el mapeado que nos darán puntos extra. Los medallones, las vasijas y los cuadros pueden ser recogidos pasando sobre ellos.



Existe otro elemento que da puntos extra, el oficial aleman, al que deberemos disparar. Los oficiales se reconocen por estar sentados en sus

mesas (en contraposición a los guardias, que son los enemigos móviles que más abundan en el juego).

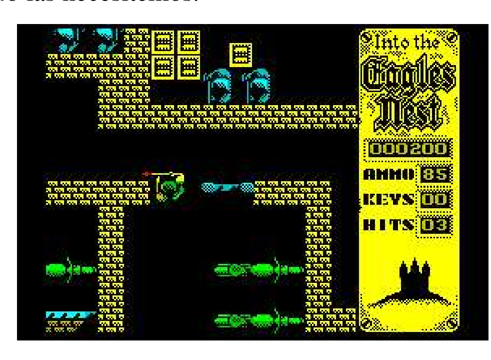




- Detonadores y Explosivos: Estos elementos nos permitirán detonar cada una de las plantas. Una vez activado un detonador (disparándole), deberemos ser rápidos para abandonar la zona antes de que explote. Cuidado con disparar a los explosivos, o moriremos de forma instantánea.
- Ascensor/escaleras: Nos permitirá movernos entre las diferentes plantas del castillo.

Consejos de juego

Hay que evitar desperdiciar munición: nunca dispares más veces de las necesarias, recuerda que basta un bala por nazi en nivel fácil o dos balas por nazi en nivel difícil. No sólo hay que tener en cuenta la cantidad de disparos para no malgastar balas, sino que además las balas no se detienen en el borde del área de visión que tenemos, y pueden impactar en objetos más alejados de nosotros (que no vemos), como la dinamita. También es importante que estés situado correctamente en la línea de disparo del enemigo si quieres eliminarlo correctamente. A la hora de recargar tu arma, no recojas municiones si todavía estás cerca del límite de 99 balas, ya que todas las que excedan de 99 se perderán. En ocasiones es mejor reservarlas para cuando las necesitemos.



Administra correctamente la comida y las medicinas, si apenas te han tocado resulta mucho más útil coger la comida (que recupera menos salud) que los botiquines (y reservar estos para niveles de salud mucho más bajos). Una manera muy eficaz de evitar que nos golpeen por un lado mientras disparamos es protegernos al lado de algún barril o pared. Especialmente útiles son las esquinas: podemos disparar utilizando las esquinas de forma que los enemigos no puedan dispararnos a nosotros; para ello en lugar de cruzar completamente la esquina, lo haremos a medias y podremos disparar a los enemigos sin que ellos nos disparen a nosotros.

Conviene tener en cuenta que cada vez que entramos en una planta se regenera la munición disponible para recoger en el mapeado, así como las llaves, pero no ocurre lo mismo en el caso de la comida, por lo que podemos estar tranquilos de que no nos quedaremos sin munición al desplazarnos entre plantas (no es necesario que dejemos reservas en un nivel para cuando volvamos a él).

Por último, en ocasiones se producen situaciones en las cuales el rehén nos impide movernos porque bloquea una salida, o es posible que incluso nos impida eliminar a soldados alemanes. En esos casos podemos dispararle, lo cual forzará a que se mueva.

Valoración del juego

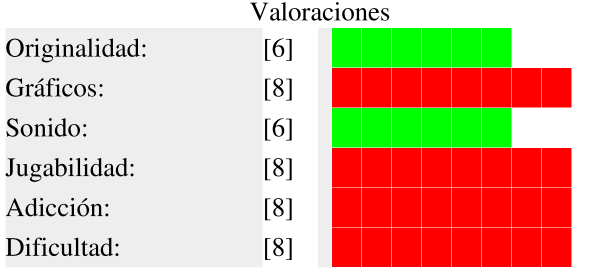
Aunque a nadie se le escapa que este juego toma muchos elementos del mítico Gauntlet (vista aérea, scroll en cuatro direcciones, gran cantidad de enemigos a abatir, items que recoger), hay que tener en cuenta que tiene una componente estratégica que no tienen otros juegos de su género. Debemos administrar lo mejor posible las llaves, volar las plantas, rescatar a los prisioneros, administrar correctamente los recursos (munición y salud), y todo esto teniendo en cuenta que posiblemente tendremos que volver a pasar por la planta del castillo en que nos encontramos en cada momento. Todo esto resulta en una gran cantidad de horas de juego, aderezado con la posibilidad de intentar batir el juego de nuevo en un nivel de dificultad más alto.

Los controles dejarán satisfechos a todo el mundo, porque aparte de poder redefinir totalmente el teclado, soporta los cursores y los joysticks Kempston y Sinclair.

Los gráficos son bastante grandes y de calidad, y el scroll no es demasiado molesto. En cuanto al sonido, simplemente puede decirse que cumple su función sin destacar especialmente (algo a lo que estamos acostumbrados en la mayoría de juegos de Spectrum).

En definitiva, un juego muy recomendable, que nos proporcionará muchas horas de diversión y un entretenido reto debido a su moderada dificultad.

Valoraciones



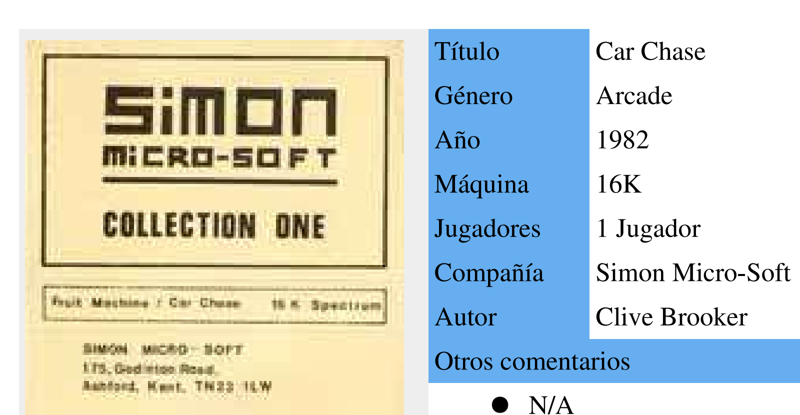
Originalidad:	[6]
Gráficos:	[8]
Sonido:	[6]
Jugabilidad:	[8]
Adicción:	[8]
Dificultad:	[8]

Trucos: Puedes encontrarlos en The Tip Shop http://www.the-tipshop.co.uk/cgi-bin/search.pl?name='Reveal'

Descárgalo de: WOS http://www.worldofspectrum.org/infoseek.cgi?regexp=^Reveal$&pub=^Mastertronic+Ltd$

SROMERO (NoP)

CAR CHASE



Título	Car Chase
Género	Arcade
Año	1982
Máquina	16K
Jugadores	1 Jugador
Compañía	Simon Micro-Soft
Autor	Clive Brooker
Otros comentarios	

- N/A

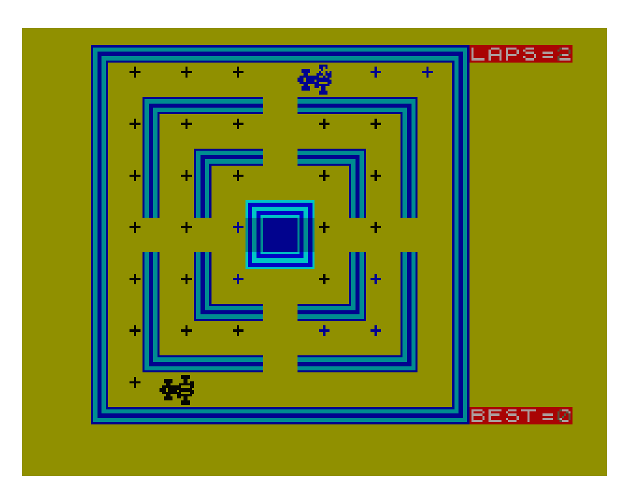
En los primeros tiempos del Spectrum, no era raro encontrar juegos comerciales hechos en BASIC. Creemos que uno de los ejemplos que a todos se nos vienen a la memoria puede ser Football Manager. Pero hubo muchos más. Hemos seleccionado este Car Chase por algún otro motivo que enseguida descubriréis.



Instrucciones

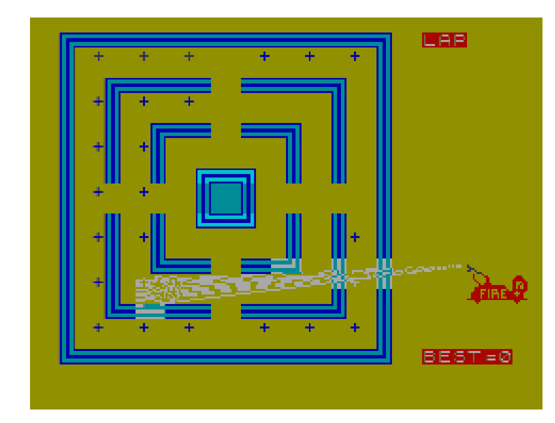
Un juego para Spectrum 16K, escrito en BASIC... La verdad es que no es ninguna maravilla técnica. Sin embargo, nos llamó poderosamente la atención su desarrollo. ¡Eureka! ¡Si es el papá de Amoto's Puf! Vaya, vaya. Y nosotros que creíamos que juntar coches y comecocos era una idea "original"...

Pues no lo es, pero es que Car Chase tampoco es original ni mucho menos. En una época en la que no existían licencias ni exclusivas, y en las que las casas de software hacían conversiones libremente de los éxitos de recreativa a los ordenadores domésticos, Car Chase retoma el desarollo de una placa de nada menos que el año 1979, llamada Crash.



Corriendo por el laberinto

Como podemos ver, el campo de juego consiste en 3 calles concéntricas con 4 lugares en los que pasar de una a otra calle. El objetivo consiste en pasar por encima de todos los puntos sin chocar con el coche contrario. Lo que parece fácil a priori no lo es tanto, ya que nuestro margen de maniobra es muy limitado.



¡Hemos chocado! Pero no hay problema, los bomberos se encargan de todo

Sólo podemos cambiar de una calle a otra adyacente en dichos 4 puntos determinados. La estrategia consiste en mantenernos lo más alejados posible del

otro coche antes de llegar a cada cruce, para tener opciones a cambiar. En la recreativa había un botón para acelerar, pero en esta versión no contaremos con esa ventaja.

El juego carga en dos veces. Tras el primer bloque nos muestra las instrucciones. Después de leerlas, cargaremos el segundo, donde está el juego en sí. La verdad es que es una buena forma de aprovechar los escasos 16KB de memoria sin descuidar una presentación elegante.

Técnicamente el juego no tiene nada destacable que decir, es correcto para estar escrito en BASIC. Es lento y los coches parpadean ostensiblemente al moverse. Ni en el sonido es original, usa la fanfarria del Moon Cresta antes de cada carrera.

Pero lo que hoy día sería un juego mediocre hasta para presentarlo al concurso de BASIC 2004, hay que verlo con la perspectiva de que han pasado 22 años. En su momento debió ser un entretenimiento bastante divertido, por tanto vamos a pasar nuestra puntuación por el tamiz de la edad, y seremos algo condescendientes con él.

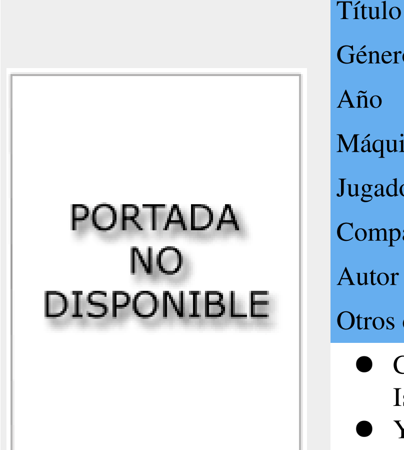
Valoraciones

Originalidad:	[4]	████
Gráficos:	[6]	██████
Sonido:	[4]	████
Jugabilidad:	[5]	█████
Adicción:	[6]	██████
Dificultad:	[7]	███████

Descárgalo de:

- WOS http://www.worldofspectrum.org /infoseek.cgi?regexp=Car+Chase

REVEAL



Título	Reveal
Género	Arcade
Año	1989
Máquina	48K
Jugadores	1 Jugador
Compañía	Mastertronic Ltd.
Autor	I. Heath

Otros comentarios

- Crash Issue 62 ftp://ftp.worldofspectrum.org/pub/sinclair/magazines/Crash/Issue62/Pages/Crash6200068.jpg
- Your Sinclair Issue 40 ftp://ftp.worldofspectrum.org/pub/sinclair/magazines/YourSinclair/Issue40/Pages/YourSinclair4000066.jpg
- Microhobby 190 ftp://ftp.worldofspectrum.org/pub/sinclair/magazines/MicroHobby/Issue190/Pages/MicroHobby19000032.jpg



Menú de opciones

Estamos ante una nueva variación de la idea clásica del comecocos. En este caso, nuestra tarea consiste en devolver la luz a un mundo tridimensional que se ha sumido en las tinieblas. Para ello, debemos pasar por encima de cada una de las casillas que componen cada pantalla. Una vez lo hayamos iluminado por completo, podremos dirigirnos a la salida hacia el siguiente nivel.

La mecánica es siempre la misma. Al comenzar se nos mostrará el nivel completamente iluminado por unos instantes, que deberemos aprovechar para intentar memorizarlo lo mejor posible. A continuación, todas las casillas se apagan, quedando iluminadas solamente aquellas que vayamos

encendiendo, y aquella sobre la que estén nuestros enemigos. Ni que decir tiene que el contacto con uno de ellos nos restará una vida.

Podremos manejar a nuestro personaje mediante el teclado (redefinible) y los ya clásicos joysticks Sinclair y Kempston.

Por cierto que, en este caso, el botón de disparo debería llamarse botón de "suicidio", ya que nos permite volver a la posición inicial perdiendo una vida. La verdad es que no tiene demasiado sentido...

Simplemente reseñar que a veces nuestros enemigos se dedicarán a hacernos la puñeta apagando casillas que hayamos encendido previamente, normalmente en el lado opuesto de la pantalla al que nos encontramos.



Primero se nos muestra el mapa completo...

¿Qué aporta este título frente a otros como puedan ser, a bote pronto, El Pintor (ya comentado en el primer número de esta revista) o Humphrey? Únicamente la inclusión de una vista isométrica en lugar de la clásica vista cenital. También hay que destacar que el movimiento es muy suave. El tratamiento del color no es nada complicado, se ha optado por usar un diseño monocromático, para evitar el típico emborrachamiento de atributos del Spectrum.

Por tanto, técnicamente se trata de un juego correcto y bien realizado, si bien para la época en que fue lanzado ya habíamos sido testigos de algunas grandes maravillas en nuestro Spectrum, por lo que no destaca en este aspecto. El sonido es casi prescindible, limitándose a los típicos ruiditos al desplazarnos por la pantalla. La melodía del menú es un clásico de Scott Joplin adaptado a las limitaciones del *speaker*.

En resumen, un juego que no pasará a la historia por destacar en ningún aspecto pero que bien puede proporcionarnos unos ratos de diversión en las gélidas tardes de invierno que se nos avecinan.✍

Valoraciones

Originalidad:	[5]	█████
Gráficos:	[7]	███████
Sonido:	[5]	█████
Jugabilidad:	[9]	█████████
Adicción:	[7]	███████
Dificultad:	[7]	███████

Trucos:

Puedes encontrarlos en The Tip Shop http://www.the-tipshop.co.uk/cgi-bin/search.pl?name='Reveal'

Descárgalo de:

- WOS http://www.worldofspectrum.org/infoseek.cgi?regexp=^Reveal$&pub=^Mastertronic+Ltd$

FALVAREZ



...y ahora nos toca a nosotros ir descubriendo las casillas



MIGUEL

Los extraordinarios casos del Dr. Van Halen: Misterio en la catedral

En el papel del Dr. Van Halen, un hombre de comienzos del siglo veinte, habitante de la ciudad de Brujas y atormentado por la misteriosa desaparición de su amada, debemos solucionar los diferentes enigmas que se nos van planteando, ayudados por nuestro rol aventurero y conocedor de lo oculto. A lo largo de nuestras andanzas se nos irá descubriendo, con cuentagotas, el estado de Helen, algo que le da un poco más de profundidad al argumento.

Nuestra aventura comienza cuando suena el timbre de la puerta de nuestra casa y nos hacen entrega de un telegrama, citándonos en una recóndita aldea de Francia llamada Le Nebleau con el padre Pierre. A partir de este momento nos veremos inmersos en una serie de acontecimientos paranormales que harán que nuestro juicio se nuble. El argumento del juego no es muy complicado y, aunque para solucionarlo no tenemos que seguir una linealidad absoluta, tampoco es demasiado complicado hacerlo sin la solución salvo, quizá, en algún punto muy determinado en el que puedes ver cómo te matan una y otra vez. Por suerte, el juego no se basa en tiempo para solucionar cualquier problema que se nos plantea y podremos dedicar todo el que deseemos para hallar el uso del objeto correcto o el movimiento preciso que nos sacarán del apuro.



Presentación

El mapeado del juego consta de 28 pantallas repartidas en dos zonas diferenciadas: la mansión donde habitamos en Brujas y la villa francesa. En ambas hay zonas secretas a las que deberemos descubrir cómo acceder, personajes con los que conversaremos y nos ayudarán u otros que intentarán que no consigamos nuestro fin. Las pantallas están repletas de diferentes objetos, algunos de ellos necesarios y otros prescindibles para terminar nuestra aventura.

Respecto a los gráficos los aficionados a este tipo de juegos sabemos que no es lo más importante, que prima más la profundidad del argumento o los diálogos con otros personajes, y en este caso no cambia. Tenemos una breve descripción en apenas media docena de localizaciones, con unos gráficos bien realizados. Y en el resto se nos describen mediante un texto lo suficientemente explícito como para situarnos en el lugar.

El sonido es prácticamente inexistente. Nos tendremos que conformar con el típico 'clic' en cada pulsación de las diferentes teclas.



Le Nebleau

La comprensión del programa a nuestras órdenes es bastante correcta. Con frases del tipo 'meter telegrama en maletin' o 'coger sombrero' el

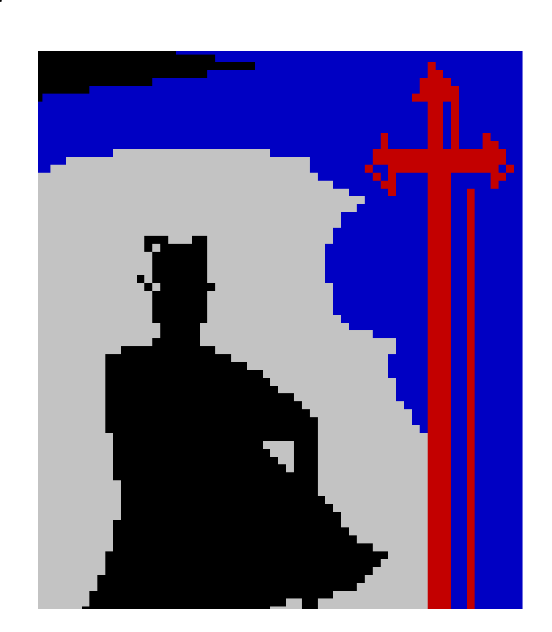
programa nos entiende a la perfección. En todo el devenir de la aventura hasta que la completamos, en pocas ocasiones respondió que no podía interpretar nuestros mandatos.

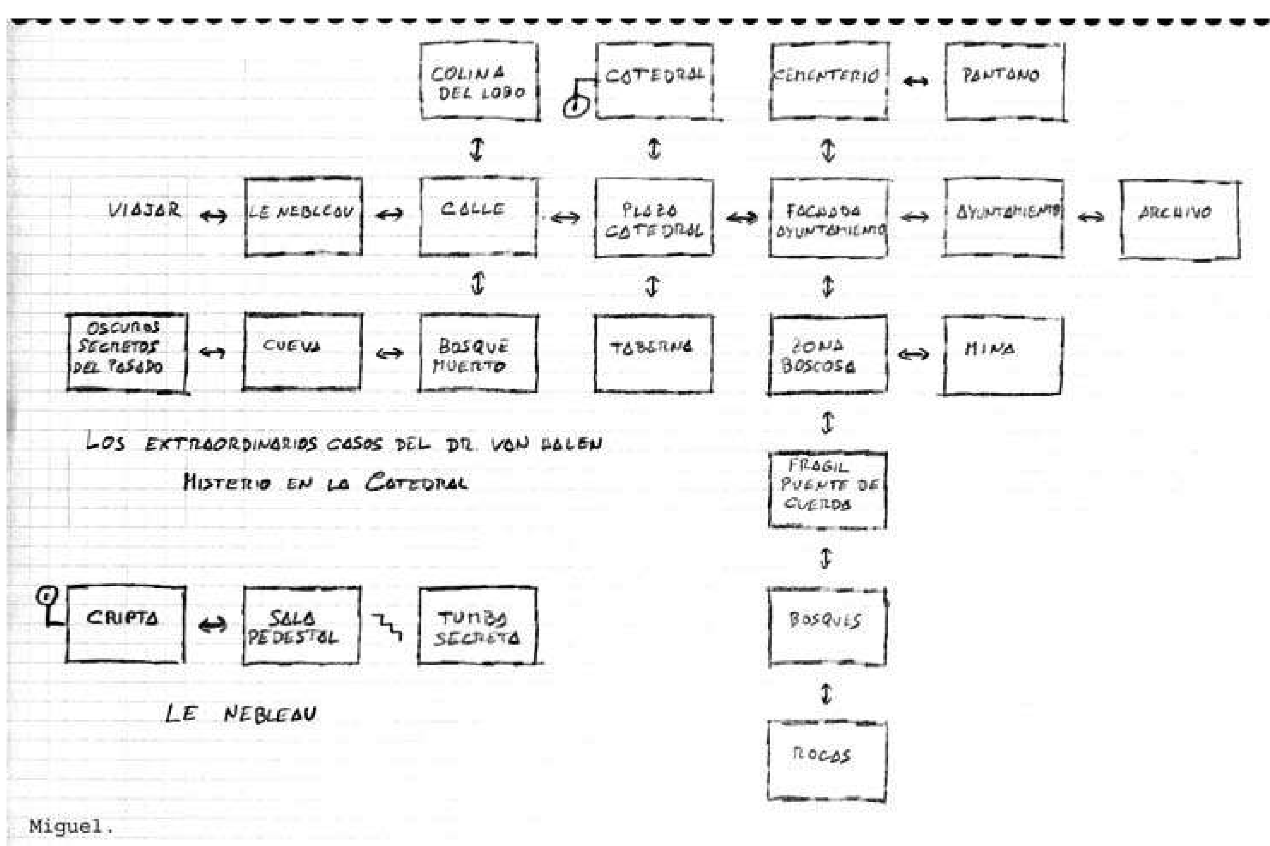
En definitiva, 'Misterio en la catedral' es un buen juego, entretenido, que para los amantes de las aventuras conversacionales puede resultar un poco sencillo de resolver, pero que en absoluto es simple, aunque esperamos que las siguientes entregas de las aventuras del Dr. Van Halen se vayan haciendo un poco más complicadas.

Como crítica en general podemos comentar que nos quejamos de que las nuevas producciones para nuestra máquina de culto son escasas. Pues parece increíble que esta aventura conversacional haya pasado prácticamente desapercibida, y más siendo gratuita, entre los usuarios del Sinclair ZX Spectrum.

Habréis podido observar que esta vez hemos puesto al descubierto un juego de una forma un tanto escueta. Y es que ya llevábamos la mitad del artículo escrito cuando nos dimos cuenta de que el propio juego adjunta la solución al mismo. Así que para no dejar de lado esta buena pieza del software actual, pero no duplicar información de manera innecesaria, hemos decidido haceros un regalo. Un mapa de la aventura, a la usanza de los *viejos tiempos*. ✍









Pedrete

Creación de ficheros de cinta TZX en GNU/Linux

Hoy en día, el sistema operativo GNU/Linux está teniendo un gran auge entre los usuarios de ordenadores personales. En este artículo se va a tratar de orientar al usuario de este sistema operativo en la manera de poder crear sus propios ficheros de cinta en formato TZX y, de esta forma, poder colaborar en la preservación de programas.

Introducción

Primero de todo, hay que decir que en este artículo se va a suponer que disponemos de un sistema operativo GNU/Linux correctamente instalado en nuestra máquina. Y que además sabemos manejarnos en él con cierta soltura: manejar comandos desde una consola, compilar un kernel, etc...

El objeto desde artículo es fomentar en lo posible el uso de herramientas nativas para GNU/Linux, pero el abanico de estas utilidades es muy escaso, aunque se puede recurrir a otros artificios como emuladores para poder usar herramientas de otros sistemas operativos de las que GNU/Linux no dispone (lo cual puede ser motivo de un futuro artículo).

Configuración del equipo

En este caso es obligado disponer de una tarjeta de sonido con entrada de línea (LINE-IN) bien configurada en nuestro equipo. Cualquier tarjeta (incluso las integradas en la placa base del ordenador) debería disponer de ella.

Además necesitaremos un reproductor de cassette (ya sea un walkman, el mítico Computone o, incluso, el propio cassette integrado del Spectrum +2A) y un cable de audio con dos jacks de 3.5mm (son como los de los auriculares de un walkman), uno en cada extremo del cable. El cable que se suministraba con el propio Spectrum 16K/48K nos puede valer perfectamente.

Un extremo de este cable se conectará a la salida EAR (u OUT) del reproductor de cassette y el otro extremo del cable deberá ir a la entrada LINE-IN (o de línea) de la tarjeta de sonido de nuestro ordenador.

Ahora queda configurar el sonido en GNU/Linux, pero antes un pequeño receso para contar brevemente algo sobre el sistema de sonido del kernel de Linux.

El kernel de Linux, actualmente en su rama 2.6, dispone de dos tipos de sistemas de sonido: OSS y ALSA. OSS es el sistema de sonido que se viene usando desde versiones anteriores del kernel y al que ya no se le da soporte en el kernel (aunque sí dispone de una versión comercial que lo hace) y se apuesta a favor del uso de ALSA.

ALSA (Advanced Linux Sound Architecture) es un sistema sonido completamente nuevo, hecho desde cero y con soporte para gran cantidad de tarjetas de sonido tanto antiguas como actuales (cosa que no se puede asegurar de OSS).

Pero lo más interesante es que ALSA tiene soporte para emular OSS, por lo que con el sistema ALSA podremos seguir usando aplicaciones que hayan sido realizadas especifícamente para OSS, como es nuestro caso. Este hecho nos va a ocurrir con algunas de estas aplicaciones por lo que es importante que si usamos ALSA, verifiquemos que disponemos de la emulación OSS, ya que sin ella las aplicaciones como MakeTZX no funcionarán pues no han sido realizadas para utilizar ALSA, sino OSS.

Para confirmarlo ejecuta en una consola de GNU/Linux el siguiente comando:

```
$ grep -i oss /usr/src/linux/.config
```

y se debería mostrar unas líneas como éstas (en algunas puede aparecer '=y' en vez de '=m' y viceversa. El caso es que no aparezca un '=n'):

```
CONFIG_SND_OSSEMUL=y
CONFIG_SND_MIXER_OSS=m
CONFIG_SND_PCM_OSS=m
CONFIG_SND_SEQUENCER_OSS=y
```

Si no, toca compilar el kernel con estas opciones modificadas y rearrancar el ordenador.

Sobra decir que si no usamos ALSA y usamos sólo OSS, no hará falta hacer ningún cambio de lo que se ha dicho anteriormente pues ya tenemos instalado el sistema de sonido nativo que usarán las aplicaciones que empleemos.

Hecho este breve paréntesis seguimos con la configuración del sonido: ahora toca ajustar el volumen. Es importante que el volumen de la entrada de línea esté activado a un nivel medio-alto, ya que suele ser bastante común olvidarse de modificar este volumen y por tanto no oiremos por los altavoces nada de lo que el ordenador esté digitalizando a fichero TZX.

Una vez hecho todo esto ya podemos probar si tenemos bien configurado todo el apartado de sonido. Pondremos una cinta cualquiera en el reproductor (recuerda subir también el volumen del reproductor), pulsaremos PLAY y ejcutaremos el siguiente comando en una consola:

```
$ rec prueba.voc
```

y una vez escuchado algo de sonido, pararemos el cassette con el STOP, pulsaremos CTRL+C en la consola para parar la grabación y reproduci-remos el ar-chivo recién grabado con:

```
$ play prueba.voc
```

Si escuchamos el sonido del contenido de la cinta podemos seguir a la siguiente sección. Si no, habrá que seguir probando. Recomiendo volver a leer esta sección revisando cada uno de los pasos realizados.

Instalación de la herramienta MakeTZX

MakeTZX es un programa desarrollado por el grupo italiano de programación RAMSOFT que nos permitirá transferir los programas desde la cinta a fichero TZX de una manera, como veremos, bastante sencilla y sin requerir apenas interacción por parte del usuario.

Para ello, nos dirigiremos hasta su página web, donde elegiremos la opción MakeTZX y buscaremos el enlace que nos permite descargar la versión GNU/Linux de esta utilidad (a día de hoy la versión 2.31 es la más reciente para el sistema operativo que nos ocupa, a pesar de estar la 2.33 tanto para DOS como para Windows. Aun así la versión de la que disponemos funciona de maravilla).

Ya sólo nos falta descomprimirlo en un directorio temporal y copiar el fichero 'maketzx' contenido en ese directorio recién creado a un directorio contenido en nuestro PATH de ejecución, por ejemplo, a /usr/local/bin. A su vez otorgaremos los permisos de ejecución y propietarios pertinentes y que creamos convenientes para poder usarlo, pe.e, como usuario y no requerir privilegios de ROOT cada vez que queramos usarlo.

Una vez hecho esto ya podremos usar el programa y lo podremos probar ejecutando simplemente:

```
$ maketzx
```

con los que nos aparecerá un listado de opciones como el de la imagen.

```
-=[ MakeTZX v2.31 ]=- (C) 1998-2001 RAMSOFT, a ZX Spectrum demogroup.

Converts sample files of ZX Spectrum tapes to the TZX format.
Supports VOC/WAV/CSW/IFF/OUT files and realtime decoding from soundcard input.

Syntax: MAKETZX inputfile [outputfile] [switches]
For more information, please read the manual carefully (MTZXMAN.HTM)

Switches: (default values are in square brackets)
  -a: enable loader type autodetect    -r: enable realtime conversion
  -l: select loading standard type         s<n>: set sampling rate [32258 Hz]
      a: Alkatraz (auto)                   t<s> or t<m:s>: set recording time
      b: Bleepload (auto)                              (m=minutes, s=seconds)
      f: Softlock (auto)                   k: save the samples to a WAV file
      n: Normal (forces ROM timings)   -f: enable digital filter
         f: short pilots detection         o<n>: set filter order [2]
      s<n>: Speedlock <n> (auto 1,2,7)     t<n>: set type (3=LP 4=BP 5=HP) [4]
      z: ZetaLoad (auto)                   h<n>: set upper cutoff freq [4100]
      o: Paul Owens (auto)                 l<n>: set lower cutoff freq [600]
      c: Activision                        p<n>: prototype, 1=Butt. 2=Cheb. [1]
      p: PowerLoad (auto)                  r<n>: set ripple (Cheb.) [1 dB]
      g[n]: GB Max Biturbo <n> (auto)      3: disable 3DNow! acceleration
  -x: display values in hex format     -b: enable TZX beautifier
  -h: disable hi-fi tones (Speedlock)  -k<n>: skip <n> blocks before loaders
akelarre@orabidoo:~$
```

Transfiriendo las cintas al ordenador

Ya mismo se podría afirmar que hemos llegado a la parte más sencilla de todo el proceso de creación de archivos TZX, pues la mayor parte de las ocasiones consistirá en ejecutar un simple comando en la consola de nuestro ordenador y de pulsar PLAY en el reproductor de cassette.

Como se ha visto en el apartado anterior, la utilidad MakeTZX dispone de una serie de opciones de las que apenas usaremos un par de ellas y las demás sólo serán necesarias en caso de complicación en el proceso de transferencia.

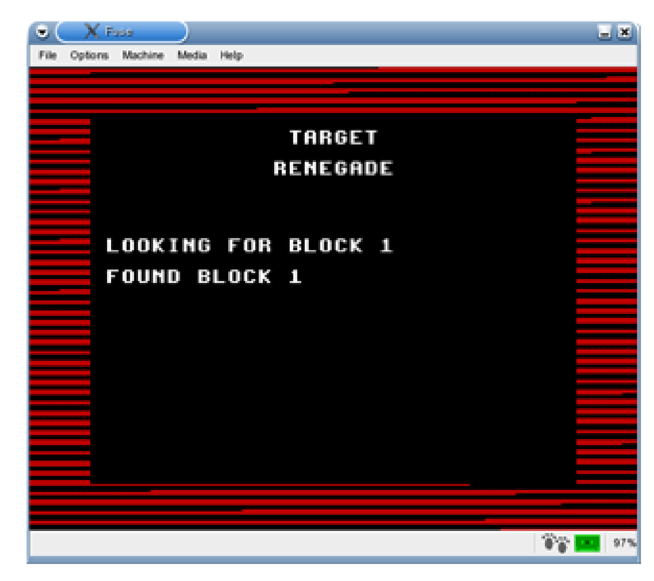
El comando que usaremos más habitualmente es el siguiente:

```
$ maketzx -a -r
```

donde con la opción '-a' indicamos al MakeTZX que autodetecte el tipo de carga del programa (cosa que hace bastante bien si la soporta), con '-r' le decimos que procese el sonido de carga en tiempo real y, finalmente, será el fichero donde queremos almacenar el TZX resultante.

Es muy importante no usar nunca jamás la opción '-b' pues esta opción crea archivos TZX imperfectos que serán directamente deshechados por las páginas que preservan estos programas.

Cuando ejecutemos ese comando y antes de poder empezar la transferencia, nos aparecerán en pantalla un par de ecualizadores, significando uno el volumen y el otro el *clipping*.



Target Renegade (Imagine Software) [target_renegade.tzx] Tipo de carga: Estándar de ROM + Custom loader

El significado del volumen es obvio y no deberá ser ni muy bajo ni muy alto, con ponerlo sobre el 75% del total es suficiente.

El *clipping* por su parte indica que la señal puede estar

```
akelarre@orabidoo:~$ maketzx -a -r tr.tzx

-=[ MakeTZX v2.31 ]=- (C) 1998-2001 RAMSOFT, a ZX Spectrum demogroup.

p CMedia PCI system driver (OSS v3.8.16) (Ramsoft 0.50)
* Sampling mode: 8-bit signed
p Sampling rate: 32258 Hz.
* Press RETURN to start conversion when done with vu-meters.
p Max recording time is 36:59:04 (4294967295 bytes)

Block  1 => Program: TARGET      - Header: Length=   17, Pause=1006ms.
Block  2 => -------------------- - Line=   10, Length=  154, Pause=4646ms.
Block  3 =>   Bytes: $           - Header: Length=   17, Pause=1008ms.
Block  4 => -------------------- - Start=16384, Length= 6914, Pause=13ms.
Block  5 =>   Bytes: c           - Header: Length=   17, Pause=1005ms.
Block  6 => -------------------- - Start=34816, Length=29442, Pause=10508ms.
Block  7 => F:136 - Speed: 146% - Chk=OK! (114), Length=    3, Pause=1ms.
Block  8 => F:136 - Speed: 146% - Chk=OK! (180), Length=15106, Pause=9677ms.
Block  9 => F:136 - Speed: 146% - Chk=OK! (116), Length=    3, Pause=1ms.
Block 10 => F:136 - Speed: 146% - Chk=OK! (102), Length=15106, Pause=9241ms.
Block 11 => F:136 - Speed: 146% - Chk=OK! (118), Length=    3, Pause=1ms.
Block 12 => F:136 - Speed: 146% - Chk=OK! (156), Length=15106, Pause=9986ms.
Block 13 => F:136 - Speed: 146% - Chk=OK! (120), Length=    3, Pause=1ms.
Block 14 => F:136 - Speed: 146% - Chk=OK! (206), Length=15106, Pause=9210ms.
Block 15 => F:136 - Speed: 146% - Chk=OK! (122), Length=    3, Pause=1ms.
Finding pause...p Recorded 19103744 samples in 09:52
Block 16 => F:136 - Speed: 146% - Chk=OK! ( 52), Length=15106, Pause=21402ms.

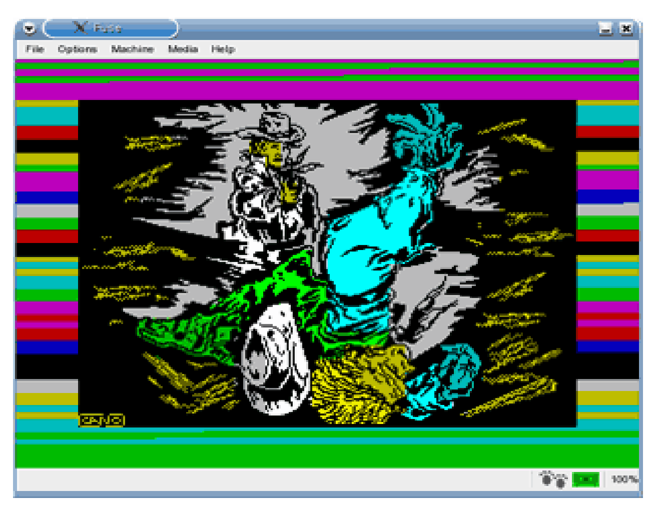
Done!

akelarre@orabidoo:~$ 
```

demasiado distorsio-nada y puede aparecer recortada con lo que sería defectuosa y, por tanto, provocar erores de carga.

Un truco consiste en pulsar PLAY en el reproductor y escuchando sonido de carga, bajar el volumen a 0 (tanto de ordenador como del reproductor) e ir incrementando ambos hasta que ambas barras de ecualización tengan unos valores aceptables. Recomiendo consultar la ayuda del propio MakeTZXdonde se explica este proceso de afinación claramente.

Una vez realizado este ajuste (no será necesario hacerlo cada vez que queramos usar MakeTZX, salvo que cambiemos los volúmenes antes mencionados), podremos pulsar y comenzar la conversión de nuestro programas.



Desperado (Topo Soft) [desperado.tzx] Tipo de carga: Estándar de la ROM

En este momento, MakeTZX se pondrá en modo de escucha y es en este momento cuando debemos pulsar PLAY en el reproductor con la cinta del programa que queremos transferir.

A medida que se oye el sonido de carga, en la pantalla iremos viendo diferentes datos según el programa se va cargando. Una vez finalizado el proceso de transferencia bastará con pulsar de nuevo la tecla con lo que el proceso se dará por finalizado y dispondremos de un flamante archivo TZX listo para ser cargado en nuestro emulador favorito.

A continuación se muestran algunos ejemplos de juegos transferidos a formato TZX, cada uno de ellos condifernetes formatos de carga para comprobar el buen funcionamiento de la utilidad MakeTZX.

Como se puede apreciar en este juego, los primeros bloques son de carga estándar de la ROM pero las fases están grabadas en un formato turbo. Además este formato turbo tiene una cabecera casi inapreciable (con pausas de apenas 1ms) pero que el MakeTZX detecta perfectamente.

Vemos en la imagen como el Target Renegade carga perfectamente y sin errores en el emulador FUSE.

En este caso podemos ver que aunque el juego cargue de un modo colorista y aparentemente no estándar, se comprueba que el modo en que fue grabado fue con la velocidad y el formato estándar de la ROM, excepto por el hecho de que el último bloque no tiene cabecera.

Y a continuación se muestra la colorista carga del Desperado que, como hemos dicho antes, es simplemente un bloque de datos estándar de la ROM pero cargado de una forma diferente

Y finalmente tenemos un caso de carga un poco más complejo. Como vemos en el proceso de carga en tiempo real del programa, el MakeTZX autodetecta que se va a cargar un programa con sistema Alkatraz y se coloca en ese modo de proceso.

```
akelarre@orabidoo:~$ maketzx -a -r desperado_a.tzx

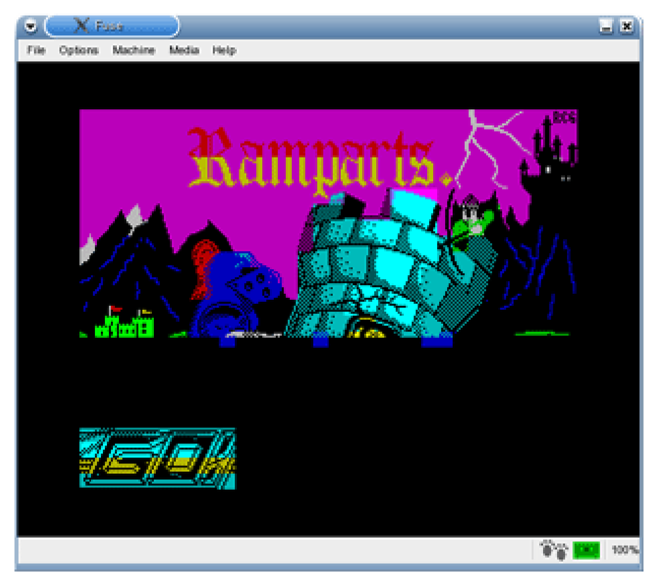
-=[ MakeTZX v2.31 ]=- (C) 1998-2001 RAMSOFT, a ZX Spectrum demogroup.

p CMedia PCI system driver (OSS v3.8.16) (Ramsoft 0.50)
* Sampling mode: 8-bit signed
p Sampling rate: 32258 Hz.
* Press RETURN to start conversion when done with vu-meters.
p Max recording time is 36:59:04 (4294967295 bytes)

Block  1 => Program: DESPERADO  - Header: Length=   17, Pause=1001ms.
Block  2 => ------------------- - Line=    0, Length= 4319, Pause=8866ms.
Block  3 => Program: desperado  - Header: Length=   17, Pause=996ms.
Block  4 => ------------------- - Line=    0, Length=  762, Pause=3675ms.
Finding pause...p Recorded 8421376 samples in 04:21
Block  5 => F:  0 - Speed: 100% - Chk=OK! (130), Length=30384, Pause=39370ms.

Done!

akelarre@orabidoo:~$ █
```



Ramparts (GO!) [ramparts.tzx] Tipo de carga: Alkatraz

También se aprecia que el último bloque del sistema Alkatraz se muestra una interrogación antes del campo Length. Esto indica que el bloque no lleva un CRC que calcular y que el MakeTZX no puede determinar si es correcto. Pero no significa necesariamente que la carga sea incorrecta.

Bastará cargarlo en un emulador y comprobarlo por nosotros mismos como hacemos se ve en la figura.

Observaciones y consejos

- La utilidad MakeTZX no puede procesar cargas muy complicadas o de las que desconoce su formato. Como por ejemplo, el sistema Poliload de juegos como Rescate Atlántida o Astro Marine Corps.
- Cualquier tutorial referente al MakeTZX nos sirve para esa versión en cualesquier sistema operativo de esta utilidad, pues todas las versiones tienen las mismas opciones y operan de igual forma.
- Si conocemos de antemano el formato de carga del programa que queremos transferir, pordemos obligarle al MakeTZX a que use ese formato con la opción '-l' seguida de la letra que corresponde al tipo de formato. En este caso, no sería necesario usar la opción '-a'.
- Los filtros digitales que incorpora MakeTZX son útiles para cintas deterioradas o cargas que se resisten, aunque no esperéis milagros. Me remito a la documentación y a los enlaces que se indican más abajo para su conocer más información sobre su uso.
- Lo repito de nuevo: nunca, nunca jamás en la vida useís la opción ('-b') de embellecer los ficheros TZX, pues no respeta la perfecta preservación de los programas al transferirlos al ordenador.

Y, finalmente, recomiendo vivamente que todos los programas que se transfieran, sean enviados a cualquier página web que se ocupe de su preservación como pueden ser El Trastero o SPA2.

LINKS

- Distribuciones de GNU/Linux:
  - http://www.debian.org
  - http://www.redhat.com
  - http://www.mandrakesoft.com
- Kernel de Linux: http://www.kernel.org
- Avanced Linux Sound Architecture (ALSA): http://www.alsa-project.org
- Página oficial de MakeTZX:http://www.ramsoft.bbk.org/maketzx.html
- Manual en inglés de MakeTZX: http://www.ramsoft.bbk.org/mtzxman.htm
- Cómo pasar cintas al PC: http://www.speccy.org/trastero/tape2tap.htm
- Esquemas de carga de programas: http://newton.sunderland.ac.uk/~specfreak/Schemes/schemes.html
- El Trastero del Spectrum: http://www.speccy.org/trastero/

SPA2: http://www.speccy.org/spa2/



FALVAREZ

Programación en Basic

Aprovechando la reciente convocatoria del concurso de BASIC 2004 de la mano de Radastan, vamos a retomar el curso que dejamos en suspenso hace más de un año. En esta entrega, aprovecharemos la disección de uno de los programas presentados en la edición del concurso del año pasado para ver cómo dibujar gráficos en movimiento y cómo efectuar la lectura del teclado para transmitir las órdenes del jugador.

El juego

El juego no es más que una versión más del clásico Snake, últimamente muy de moda gracias a su inclusión en los teléfonos móviles. Es un juego muy sencillo que nos va a servir para comprender la estructura general de cualquier juego y tocar temas como lectura del teclado, impresión de gráficos en movimiento y detección de choques.

El juego está planteado en modo texto, tal y como disponían las normas del concurso, y sólo se usan un par de GDUs, para representar la cabeza de la serpiente y las frutas.

Vamos a dejar aparte todo el tema de presentación, opciones, redefinición de teclas y demás aderezos y nos vamos a centrar en lo que es el núcleo del programa, que seguirá este esquema. Al final del artículo podréis encontrar el código fuente completo.

```
REPETIR MIENTRAS la serpiente esté viva
    Mover serpiente
    Si hay colisión:
        - colisión con fruta :
            * La serpiente crece
            * Dibujamos una nueva fruta
        - colisión con pared :
            * La serpiente muere
        - colisión con ella misma :
            * La serpiente muere
    Leer teclado
```

El teclado se puede leer antes o después de mover la serpiente, la verdad es que no notaremos diferencia debido a la velocidad a la que se ejecuta el programa (sí, incluso en BASIC). Lo único diferente que notaremos será que el primer movimiento de la serpiente siempre será hacia la derecha desde su posición inicial, ya que lo habremos definido así en las condiciones iniciales.

Pintando la serpiente

La serpiente tiene una cabeza (un carácter) y una cola (compuesta por varios caracteres). La cola de la serpiente siempre va siguiendo el movimiento de la cabeza.

Una primera aproximación que se nos podría ocurrir sería calcular hacia dónde se mueve la cabeza y, a continuación, redibujar toda la serpiente. Esa solución es inaceptable por la cantidad de proceso que requeriría, y más hablando de un lenguaje interpretado, como BASIC, corriendo sobre un microprocesador a algo más de 3MHz. Debemos plantear la solución de otra manera.

Si nos fijamos bien, veremos que sólo necesitamos mover la cabeza y el último elemento de la cola. Esto es así debido a que todos los componentes de la cola son iguales gráficamente hablando. Por tanto, deberemos hacer lo siguiente:

```
    Dibujar la nueva cabeza en la posición
correspondiente
    Sustituir la antigua cabeza por un
elemento de la cola
    Borrar el último elemento de la cola
```

Para conseguir modelizar este comportamiento de manera sencilla vamos a usar unas cuantas variables auxiliares. Por un lado necesitamos conocer dónde está la cabeza y dónde el último elemento de la cola de la serpiente. Por otro lado, para evitarnos tener que andar recalculando cada vez, almacenaremos para cada elemento de la serpiente, dónde está el elemento que la precede (de esta forma, al borrar el último elemento de la cola sabremos quién pasa a ocupar ese último lugar). Este dato lo almacenaremos en las matrices x e y.

Las orientaciones las hemos codificado de la siguiente manera (a estas alturas ya os habréis dado cuenta de que lo que se le da bien al Spectrum, como cualquier ordenador, es trabajar con números):

```
eje horizontal (matriz x):
     1 - derecha
    -1 - izquierda
```

```
eje vertical (matriz y):
    1 - abajo
   -1 - arriba
```

Estos valores no los hemos determinado así al azar, sino debido a que el origen de coordenadas de la pantalla del Spectrum en modo texto, el punto (0,0) se encuentra en la esquina superior izquierda de la pantalla.

Además, usamos un par de variables, orientacionx y orientaciony, que son las que modificaremos al leer el teclado, y que indican hacia qué dirección debe moverse la cabeza de la serpiente en la siguiente iteración.

Por último, en la matriz p almacenaremos una representación abstracta de lo que vemos en pantalla, de manera que el tratamiento de colisiones sea muy sencillo. Para ello, en cada posición almacenaremos uno de los siguientes valores:

```
    0 - posición vacía
    1 - fruta
    2 - cabeza
    3 - elemento de la cola
    4 - pared
```

Éste es el código de definición de variables:

```
100  REM Definicion de variables
110  LET cabezax = 11 : REM coordenada x de la cabeza
120  LET cabezay = 5 : REM coordenada y de la cabeza
130  LET colax = 5 : REM coordenada x de la cola
140  LET colay = 5 : REM coordenada y de la cola
150  LET orientacionx = 1
160  LET orientaciony = 0
170  DIM p(23,34) : REM Pantalla
180  DIM x(23,34) : REM Orientacionesx
190  DIM y(23,34) : REM Orientacionesy
```

Partimos de que la serpiente ya está pintada. Así que, para empezar, cambiaremos la antigua cabeza por un elemento de cola:

```
3000 REM Movemos la serpiente
3005 INK 0
3010 REM Cambiamos la orientacion
3015 LET x(cabezay+2,cabezax+2) = orientacionx
3020 LET y(cabezay+2,cabezax+2) = orientaciony
3025 REM Borramos la antigua cabeza
3030 PRINT AT cabezay,cabezax ; "O"
3035 LET p(cabezay+2,cabezax+2) = 3
3040 LET cabezax = cabezax + orientacionx
3045 LET cabezay = cabezay + orientaciony
```

A continuación habría que pintar la cabeza en su nueva posición pero, ¿qué pasa si esa posición ya está ocupada? Antes de proseguir pintando, debemos ver qué hay donde vamos a dibujar.

Detección de colisiones

Al mover la cabeza de la serpiente a su nueva posición, pueden ocurrir 3 cosas:

- Que la posición de destino esté vacía.
- Que la posición de destino esté ocupada por una fruta.
- Que la posición de destino esté ocupada por la pared o la cola de la serpiente.

En el primero de los casos no haremos nada, simplemente proseguiremos con el redibujado de la serpiente.

En el segundo caso, la cola de la serpiente crecerá. Para simular este efecto, basta con no borrar el último elemento de la cola (por eso hemos decidido retrasar el borrado hasta después de la detección de colisiones.

En el último caso, se acaba la partida.

Aquí está el fragmento de código encargado de la detección de colisiones. Basta con consultar nuestra representación matricial de lo que hay en pantalla. Si en la nueva posición hallamos un 1 en la matriz, se trata de una fruta, con lo que sumamos la puntuación y generamos una nueva fruta. Si hallamos un valor mayor que uno (cola o pared), significará el final de la partida (el código lo hemos colocado a partir de la línea 9900).

```
3040 LET cabezax = cabezax + orientacionx
3045 LET cabezay = cabezay + orientaciony
3050 IF p(cabezay+2,cabezax+2) > 1 THEN GO TO 9900
3051 IF p(cabezay+2,cabezax+2) = 1
     THEN LET puntos = puntos + 10 : PRINT AT 21,10 ;
     PAPER 1 ; INK 7 ; puntos : LET comido = 1 : GO SUB 8000
```

Sumamos la puntuación correspondiente, actualizamos el marcador y anotamos en la variable comido que hemos ingerido una fruta. Esto es importante a la hora de decidir si borramos la última posición de la cola o la serpiente debe crecer.

Y en la línea 8000 hemos colocado la subrutina que se encarga de generar una nueva fruta. En la línea 8030 nos estamos asegurando de que colocaremos la fruta en un lugar vacío.

```
8000 REM Generacion de frutas
8010 LET frutax = INT(RND*30)+1
8020 LET frutay = INT(RND*20)+1
8030 IF p(frutay+2,frutax+2) = 0 THEN GO TO 8050
8040 GO TO 8010
8050 PRINT AT frutay,frutax ; INK 2 ; "{F}"
8060 LET p(frutay+2,frutax+2) = 1
8070 RETURN
```

Bien, recordemos que dejamos la serpiente a medio pintar. Debemos pintar la cabeza en su nueva ubicación, ahora que sabemos que no ha ocurrido nada grave.

```
3055 REM Pintamos la nueva cabeza
3060 PRINT AT cabezay,cabezax ; "{S}"
3065 LET p(cabezay+2,cabezax+2) = 2
```

Por último, para acabar con el movimiento de la serpiente, si no ha habido ninguna colisión, recordemos que debemos borrar el último elemento de la cola. Esto lo hace la siguiente porción de código y la subrutina a la que invoca:

```
3070 IF comido = 0 THEN GO SUB 8100
3080 LET comido = 0
...
8100 REM Borramos la cola
8110 PRINT AT colay,colax ; " "
8120 LET nuevacolax = colax + x
(colay+2,colax+2)
8130 LET nuevacolay = colay + y
(colay+2,colax+2)
8140 LET p(colay+2,colax+2) = 0
8150 LET x(colay+2,colax+2) = 0
8160 LET y(colay+2,colax+2) = 0
8170 LET colax = nuevacolax
8180 LET colay = nuevacolay
8190 RETURN
```

Lectura del teclado

La lectura del teclado la haremos mediante el uso de la sentencia INKEY$. INKEY$ devuelve el valor del carácter producido por la tecla presionada en ese momento. Es importante tener en cuenta que si, en el momento de ejecutarse la sentencia no se está pulsando ninguna tecla, INKEY$ devolverá la cadena vacía ("") como resultado. Por lo tanto, normalmente INKEY$ se incluye dentro de un bucle que espera a la pulsación de una tecla. En nuestro caso, en ese bucle moveremos la serpiente, que sigue moviéndose en una dirección aunque no pulsemos ninguna tecla.

Por tanto, lo único que haremos al detectar la pulsación de una tecla será actualizar la orientación de la serpiente, para que se gire, si procede, en la siguiente iteración del bucle.

```
3210 LET a$ = INKEY$
3220 IF orientacionx < 1 AND (a$ = "O" OR
a$ = "o")
     THEN LET orientacionx = -1 : LET
orientaciony = 0
3230 IF orientacionx > -1 AND (a$ = "P" OR
a$ = "p")
     THEN LET orientacionx = 1 : LET
orientaciony = 0
3240 IF orientaciony < 1 AND (a$ = "Q" OR
a$ = "q")
     THEN LET orientacionx = 0 : LET
orientaciony = -1
3250 IF orientaciony > -1 AND (a$ = "A" OR
a$ = "a")
     THEN LET orientacionx = 0 : LET
orientaciony = 1
```

Resumiendo

En este artículo hemos puesto en vuestras manos la base de funcionamiento de cualquier videojuego. Evidentemente, el esquema del ZXSnake se puede complicar muchísimo más. Hemos aprendido a pintar (pintar con el mínimo esfuerzo para obtener el mejor resultado), a detectar colisiones entre objetos (es decir, que los objetos interactúen), y a leer el teclado para que el usuario pueda introducir órdenes al juego.

Quizás la idea principal que se puede sacar es que resulta de gran utilidad mantener una representación abstracta de lo que estamos dibujando en pantalla a la hora de gestionar todos los elementos que componen el juego, esto es, separar la lógica del juego de su representación en pantalla. También que el proceso de dibujado suele ser el más costoso en términos de potencia de proceso, así que cuanto menos dibujemos, el juego se ejecutará de forma más fluida.

Por último, sentíos libres de destripar a fondo el código y jugar con él. No hay una única forma de hacer las cosas, y seguro que encontráis soluciones mejores a las aquí expuestas a la hora de programar un ZXSnake.✍

LINKS

- Archivos fuente del ejemplo propuesto (zxsnake.zip) http://www.speccy.org/magazinezx/revistas/10/src/zxsnake.zip

Listado completo del programa

```
1   REM *****************************************************************
2   REM ZXSnake by Federico J. Alvarez Valero (05-02-2003)
10  REM This program is free software; you can redistribute it and/or modify
11  REM it under the terms of the GNU General Public License as published by
12  REM the Free Software Foundation; either version 2 of the License, or
13  REM (at your option) any later version.
14  REM This program is distributed in the hope that it will be useful,
15  REM but WITHOUT ANY WARRANTY; without even the implied warranty of
16  REM MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
17  REM GNU General Public License for more details.
18  REM You should have received a copy of the GNU General Public License
19  REM along with this program; if not, write to the Free Software
20  REM Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
40  REM *****************************************************************

50    BORDER 7 : PAPER 7 : INK 0 : CLS
51    PRINT AT 3,13 ; PAPER 1 ; INK 7 ; "ZXSnake"
52    PRINT AT 5,9 ; PAPER 7 ; INK 0 ; "Q - ARRIBA"
53    PRINT AT 6,9 ; PAPER 7 ; INK 0 ; "A - ABAJO"
54    PRINT AT 7,9 ; PAPER 7 ; INK 0 ; "O - IZQUIERDA"
55    PRINT AT 8,9 ; PAPER 7 ; INK 0 ; "P - DERECHA"
56    PRINT AT 10,3 ; PAPER 7 ; INK 0 ; "Recoge la mayor cantidad de"
57    PRINT AT 11,3 ; PAPER 7 ; INK 0 ; "frutas posible y crece"
58    PRINT AT 12,3 ; PAPER 7 ; INK 0 ; "sin chocarte..."
59    PRINT AT 15,3 ; PAPER 7 ; INK 0 ; "Pulsa una tecla para jugar"
60    LET j$ = INKEY$
```

```
61    IF j$ = "" THEN GO TO 60

70    REM UDG
71    RESTORE 75
72    FOR i = 0 TO 7: READ d : POKE USR "S"+i,d : NEXT i
73    FOR i = 0 TO 7: READ d : POKE USR "F"+i,d : NEXT i
75    DATA 60, 66, 129, 129, 129, 129, 66, 60 : REM serpiente (S)
76    DATA 24, 60, 60, 60, 126, 251, 247, 126 : REM fruta (F)

100   REM Definicion de variables
110   LET cabezax = 11 : REM coordenada x de la cabeza
120   LET cabezay = 5 : REM coordenada y de la cabeza
130   LET colax = 5 : REM coordenada x de la cola
140   LET colay = 5 : REM coordenada y de la cola
150   LET orientacionx = 1
160   LET orientaciony = 0
170   DIM p(23,34) : REM Pantalla
180   DIM x(23,34) : REM Orientacionesx
190   DIM y(23,34) : REM Orientacionesy
200   LET puntos = 0
210   LET comido = 0
220   LET maxx = 33
230   LET maxy = 22
240   LET minx = 0
250   LET miny = 0

1000 REM Inicializacion de la pantalla
1010 BORDER 1
1015 CLS
1020 PRINT AT 21,0 ; PAPER 1 ; INK 7 ; " PUNTOS :                              "
1030 FOR c = minx TO maxx
1040 LET p(miny+1,c+1) = 4
1050 LET p(maxy+1,c+1) = 4
1060 NEXT c
1070 FOR f = miny TO maxy
1080 LET p(f+1,minx+1) = 4
1090 LET p(f+1,maxx+1) = 4
1100 NEXT f

1500 GO SUB 8000 : REM Generar la primera fruta

2000 REM Pintamos la serpiente (posicion inicial)
2001 PAPER 7 : INK 0
2005 REM Pintamos el cuerpo
2010 FOR c = colax TO cabezax-1
2020 PRINT AT colay,c ; INK 0 ; "O"
2025 LET p(colay+2,c+2) = 3
2026 LET x(colay+2,c+2) = 1
2027 LET y(colay+2,c+2) = 0
2030 NEXT c
2040 REM Pintamos la cabeza
2050 PRINT AT cabezay,cabezax ; INK 0 ; "{S}"
2055 LET p(cabezay+2,cabezax+2) = 2
2056 LET x(cabezay+2,cabezax+2) = 1
2057 LET y(cabezay+2,cabezax+2) = 0

3000 REM Movemos la serpiente
3005 INK 0
3010 REM Cambiamos la orientacion
3015 LET x(cabezay+2,cabezax+2) = orientacionx
3020 LET y(cabezay+2,cabezax+2) = orientaciony
3025 REM Borramos la antigua cabeza
3030 PRINT AT cabezay,cabezax ; "O"
3035 LET p(cabezay+2,cabezax+2) = 3
3040 LET cabezax = cabezax + orientacionx
3045 LET cabezay = cabezay + orientaciony
3050 IF p(cabezay+2,cabezax+2) > 1 THEN GO TO 9900
3051 IF p(cabezay+2,cabezax+2) = 1
     THEN LET puntos = puntos + 10 : PRINT AT 21,10 ;
     PAPER 1 ; INK 7 ; puntos : LET comido = 1 : GO SUB 8000
3055 REM Pintamos la nueva cabeza
3060 PRINT AT cabezay,cabezax ; "{S}"
3065 LET p(cabezay+2,cabezax+2) = 2
3070 IF comido = 0 THEN GO SUB 8100
3080 LET comido = 0

3200 REM Leemos el teclado
3210 LET a$ = INKEY$
3220 IF orientacionx < 1 AND (a$ = "O" OR a$ = "o")
     THEN LET orientacionx = -1 : LET orientaciony = 0
3230 IF orientacionx > -1 AND (a$ = "P" OR a$ = "p")
     THEN LET orientacionx = 1 : LET orientaciony = 0
3240 IF orientaciony < 1 AND (a$ = "Q" OR a$ = "q")
     THEN LET orientacionx = 0 : LET orientaciony = -1
3250 IF orientaciony > -1 AND (a$ = "A" OR a$ = "a")
     THEN LET orientacionx = 0 : LET orientaciony = 1

7998 GO TO 3000

8000 REM Generacion de frutas
8010 LET frutax = INT(RND*30)+1
8020 LET frutay = INT(RND*20)+1
8030 IF p(frutay+2,frutax+2) = 0 THEN GO TO 8050
8040 GO TO 8010
8050 PRINT AT frutay,frutax ; INK 2 ; "{F}"
8060 LET p(frutay+2,frutax+2) = 1
8070 RETURN

8100 REM Borramos la cola
8110 PRINT AT colay,colax ; " "
8120 LET nuevacolax = colax + x(colay+2,colax+2)
8130 LET nuevacolay = colay + y(colay+2,colax+2)
8140 LET p(colay+2,colax+2) = 0
8150 LET x(colay+2,colax+2) = 0
8160 LET y(colay+2,colax+2) = 0
8170 LET colax = nuevacolax
8180 LET colay = nuevacolay
8190 RETURN

9900 REM Fin de la partida
9910 PRINT AT 10,12 ; INK 2 ; "SE ACABO..."
9920 PRINT AT 11,10 ; INK 2 ; "PUNTUACION : " ; puntos
9930 PRINT AT 13,10 ; INK 0 ; "Pulsa una tecla"
9931 REM Pausa obligada para que se vean los letreros
9932 FOR i = 0 TO 100
9933 NEXT i
9940 LET i$ = INKEY$
9950 IF i$ <> "" THEN GO TO 100
9960 GO TO 9940
```



Sromero (NOP)

EL JUEGO DE LA VIDA DE JOHN CONWAY

Hasta el momento hemos aprendido a instalar y compilar programas con z88dk, utilizando para ellos ejemplos prácticos basados en la creación de una Aventura Conversacional de Texto. En nuestros anteriores números, y gracias a los ejemplos de aventuras conversacionales, hemos descubierto cómo z88dk nos permite utilizar las diferentes funciones estándar de ANSI C para realizar nuestros pequeños programas.

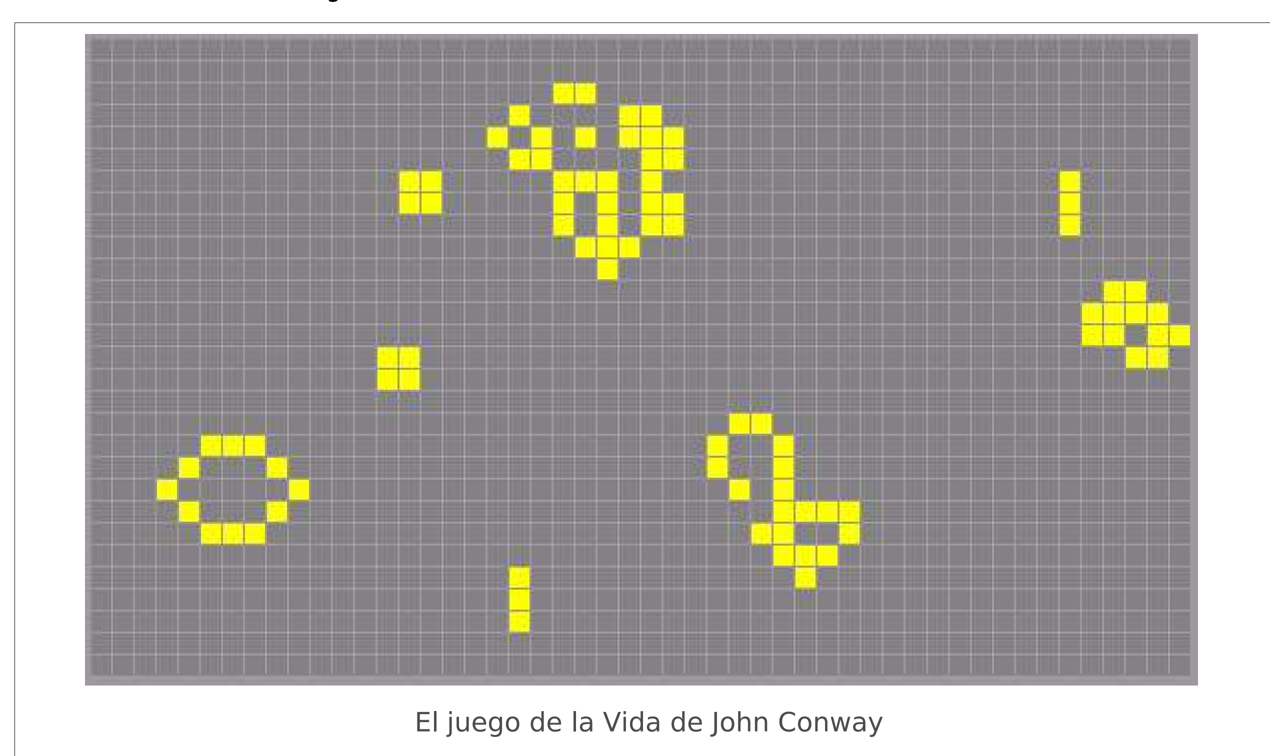
El Juego de la Vida

Esta entrega del curso de z88dk se basará en un ejemplo práctico completo comentado: la implementación para z88dk del clásico "Juego de la Vida" de John Conway. Conway fue un matemático de la Universidad de Cambridge que en 1970 inventó un sencillo juego no interactivo que permitía simular un entorno de vida basado en células individuales que morían o se reproducían mediante unas sencillas reglas.

La mejor definición del Juego de la Vida de John Conway la podemos encontrar en la Wikipedia:

El juego de la vida es un autómata celular diseñado por el matemático británico John Horton Conway en 1970. Es el mejor ejemplo de un autómata celular. Hizo su primera aparición pública en el número de octubre de 1970 de la revista Scientific American, en la columna de juegos matemáticos de Martin Gardner. Desde un punto de vista teórico, es interesante porque es equivalente a una máquina universal de Turing, es decir, todo lo que se puede computar algorítmicamente se puede computar en el juego de la vida.

Desde su publicación, ha atraído mucho interés debido a la gran variabilidad de la evolución de los patrones. La vida es un ejemplo de emergencia y autoorganización. Es interesante para los científicos, matemáticos, economistas y otros observar cómo patrones complejos pueden provenir de la implementación de reglas muy sencillas.



El juego de la Vida de John Conway

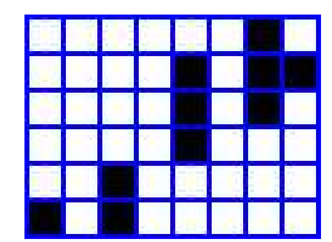
La vida tiene una variedad de patrones reconocidos

que provienen de determinadas posiciones iniciales. Poco después de la publicación, se descubrieron el pentominó R y el planeador (en inglés glider), lo que atrajo un mayor interés hacia el juego. Contribuyó a su popularidad el hecho de que se publicó justo cuando se estaba lanzando al mercado una nueva generación de miniordenadores baratos, lo que significaba que se podía jugar durante horas en máquinas que, por otro lado, no se utilizarían por la noche. Para muchos aficionados, el juego de la vida sólo era un desafío de programación y una manera divertida de usar ciclos de la CPU. Para otros, sin embargo, el juego adquirió más connotaciones filosóficas. Desarrolló un seguimiento casi fanático a lo largo de los años 1970 hasta mediados de los 80.

Cabe decir que el "Juego de la Vida" causó furor en 1970, hasta el punto en que se convirtió en el divertimento de muchos "hackers programadores" la implementación del Juego de la Vida en los potentes mainframes de Universidades y Centros de Cálculo. Hoy en día sigue siendo un buen ejercicio de programación para aquellos que empiezan a dar clases de Informática, ya que la programación es muy sencilla y no requiere grandes conocimientos del lenguaje de programación utilizado.

Reglas básicas de El Juego de la Vida

El Juego de la Vida se basa en una matriz de un tamaño determinado (como por ejemplo, 50x50, ó 32x32), que podríamos considerar nuestro "caldo de cultivo", en la cual mueren y se crean células. De forma efectiva, una célula es un 1 en una posición determinada de la cuadrícula mientras que un "espacio vacío" se representa mediante un cero.



Células vivas en la cuadrícula

El Juego de la Vida no requiere interacción por parte del usuario: a partir de un estado inicial (células diseminadas por el caldo de cultivo) se aplican una serie de reglas y se obtiene una nueva generación de células en dicho "caldo". Esta nueva generación será la entrada para volver a aplicar las reglas, y así sucesivamente.

Las reglas para cada una de las "celdillas" de nuestro caldo de cultivo son:

- Si una celdilla está ocupada por una célula y tiene una sola célula vecina o ninguna (se consideran células vecina aquellas que están alrededor de ella, en cualquiera de las 8 casillas posibles que la rodean), esa célula muere por soledad.
- Si una celdilla está ocupada por una célula y tiene 4 o más células vecinas, muere por superpoblación.
- Si una celdilla está ocupada por una célula y tiene 2 ó 3 células vecinas, sobrevive a la siguiente generación.
- Si una celdilla está vacía (no está ocupada por una célula) y tiene 3 células vecinas, nace una nueva célula en su lugar para la siguiente generación.

Con estas sencillas reglas se obtienen unos resultados sorprendentes, ya que aparecen patrones de evolución que se cambian, realizan formas y caminos determinados, etc. Así, aparece el "planeador o caminador" (un conjunto de células que se desplazan), el "explosionador" (conjunto de células que parecen formar la onda expansiva de una explosión), etc.

Implementación del Juego de la Vida en el Spectrum

Antes de codificar propiamente en C lo que es el "juego", veamos cómo sería el pseudocódigo que implementaría el diseño definido por Conway:

```
tablero[ANCHO][ALTO]

Programa Principal:
   Crear_Generación_Aleatoria()
  repetir:
     Dibujar_Generación_Actual()
     Calcular_Siguiente_Generación()
     si se pulsa la tecla 'r':
         Crear_Generación_Aleatoria()
  fin repetir
```

El anterior sería el esqueleto de la función principal, que como podréis ver en el código, se corresponde con la siguiente función main():

```
//--- Funcion principal main() --------------------------
int main( void )
{
  int i;

  GenLife();

  while(1)
  {
    DrawLife();
    Life();
    if( getk() == 'r' )
      GenLife();
  }

  return(0);
}
```

Con esto, el programa principal realizaría la impresión en pantalla de la generación actual de células (para que podamos ver la evolución visualmente) mediante la función Dibujar_Generación_Actual(). Tras esto, se calcularía la siguiente generación de células aplicando las reglas anteriormente explicadas, dentro de la función Calcular_Siguiente_Generacion() . Si en cualquier momento se pulsa la tecla 'r' (en nuestro programa) se modificará de nuevo el tablero aleatoriamente para añadir nuevas células

y que en el siguiente paso del bucle comience de nuevo la simulación.

```
Función Crear_Generacion_Aleatoria()
```

La función Crear_Generación_Aleatoria() se encargaría de vaciar el "tablero de juego" o "caldo de cultivo" (poner todos sus elementos a cero), y rellenar algunas celdillas aleatorias con células (valores 1). Dicha función la hemos definido en pseudocódigo de la siguiente forma:

```
Crear_Generacion_Aleatoria:
   Para todo 'x' y todo 'y':
     tablero[x][y] = 0
     tablero_temporal[x][y] = 0

   Repetir NUM_CELULA veces:
     xcel = x_aleatoria()
     ycel = y_aleatoria()
     tablero_temporal[ xcel ][ ycel ] = 1
```

En nuestro pseudocódigo utilizamos una matriz [ancho*alto] para representar el caldo de cultivo. Dentro de esta matriz, cada posición matriz[x][y] puede contener o no una célula mediante los valores de 0 y 1 respectivamente. Así, en un tablero de 32x32, podemos poner una célula justo en la mitad del tablero ejecutando "matriz[16][16] = 1".

```
    dimensiones:       ancho = 32
                       alto  = 16
    Tablero:           mapa[ancho][alto]
    Temporal:          temp[ancho][alto]
    Crear célula:      mapa[x][y] = 1
    Borrar célula:     mapa[x][y] = 0
```

En el caso de z88dk, en lugar de utilizar matrices bidimensionales del tipo [ancho][alto] necesitaremos utilizar un vector de tamaño [ancho*alto], ya que la versión actual de z88dk no soporta el primer tipo de matrices.

De esta forma, ahora los accesos al mapa de células quedarían así:

```
    Tablero:         mapa[ancho*alto]
    Temporal:        temp[ancho*alto]
    Crear célula:    mapa[(y*ancho) + x] = 1
    Borrar célula:   mapa[(y*ancho) + x] = 0
```

Es decir, que un array unidimensional se puede tratar como un array bidimensional donde cada una de las líneas se coloca a continuación de la anterior, y acceder a cada uno de sus elementos mediante "posición = (y * ancho_fila) + x".

Para facilitar el tratamiento de los datos, en el código definimos los siguientes macros o #defines (que hacen las veces de funciones, pero que en lugar de ser llamadas son "incluidas", evitando un CALL con sus PUSHes, POPs y sus RETs):

```
#define Celula(x,y)     (mapa[((y)*32)+(x)])
#define TempCelula(x,y) (temp[((y)*32)+(x)])
```

De esta forma en nuestro código podemos hacer simplemente:

```
 valor = Celula( 10, 10 );
 Celula( 11, 12 ) = 1;
```

Y al compilar, este código será sustituído por:

```
 valor = (mapa[((10)*32)+(10)]);
 (mapa[((12)*32)+(11)]) = 1;
```

Obviamente el código utilizando nuestros #defines es mucho más claro y más fácil de mantener. Si encontramos una manera más óptima de acceder a las células, sólo hará falta modificarlo en el #define para que al recompilar, el cambio se aplique en todo el código fuente (en lugar de tener que ir cambiándolo llamada a llamada, como nos ocurriría en caso de no haber usado #defines). Y como ejemplo de "mejora" de nuestro define, vamos a acelerar la velocidad del cálculo de la posición del array cambiando la multiplicación por 32 por un desplazamiento de bits a la izquierda, ya que en un número almacenado de forma binaria, multiplicar por una potencia n-sima de 2 equivale a desplazar n veces a la izquierda el número que queríamos multiplicar. De esta, forma como 32 es 2 elevado a la quinta potencia, nos queda que:

x * 32 = x << 5

Reemplazando esto en nuestro #define:

```
#define Celula(x,y)     (mapa[((y)<<5)+(x)])
#define TempCelula(x,y) (temp[((y)<<5)+(x)])
```

Se deja al lector como ejercicio realizar la prueba de reemplazar <<5 por *32 en el código fuente y verificar que efectivamente, el desplazamiento de bits es mucho más rápido que la multiplicación (puede apreciarse visiblemente en la velocidad de la simulación).

El código final correspondiente para nuestra función de generación aleatoria de estados iniciales es el siguiente:

```
//--- Rellenar el tablero con valores aleat. -------------
void GenLife( void )
{
   int x, y, i;

   // Inicializar la semilla de numeros aleatorios
   srand(clock());
   BORDER(0);
   CLS(0);
   printf( "\x1B[%u;%uH",(21),(1));
   printf("   ZX-Life - MagazineZX - z88dk     ");
   printf(" (r) = nueva generacion aleatoria   ");

   // limpiamos el tablero de celulas
   for( i=0; i< NUM_CELULAS; i++)
   {
      x = (rand() % (ANCHO-2)) +1;
      y = (rand() % (ALTO-2)) +1;
      TempCelula(x,y) = Celula(x,y) = 1;
   }

}
```

Como cosas que destacar de esta función tenemos:

- **srand(clock())** : Inicializa la semilla de números aleatorios usando como base el reloj del Spectrum (por reloj del Spectrum consideramos el contador que tiene el sistema y que contabiliza el número de segundos transcurridos desde el inicio del ordenador). Esto asegura que cada vez que ejecutemos el programa (en un Spectrum real o un emulador que no tenga carga automática del .tap al abrirlo con él), tengamos una secuencia de números aleatorios diferentes y no

obtengamos siempre la misma generación inicial de células. Los números aleatorios los obtendremos posteriormente con rand().

- **BORDER() y CLS()**: Estas 2 funciones cambian el color del borde y borran la pantalla respectivamente. Están implementadas en ensamblador, como puede verse en el código fuente de ZXlife . La primera cambia el borde mediante un OUT del valor del borde en el puerto que se utiliza para ello, mientras que la segunda utiliza LDIR para borrar la zona de pantalla de memoria. Este ejemplo nos permite ver lo sencillo que es integrar ensamblador en z88dk, embebiendo el código ASM dentro del propio programa.
- **printf("\x1B[%u;%uH",(21),(1))**: Este comando es el equivalente ANSI de gotoxy (x,y), es decir, posiciona el cursor en la posición (1,21) de pantalla para que el próximo printf comience a trazar las letras en esa posición. En este caso lo utilizamos para posicionar en cursor en la parte baja de la pantalla, donde escribimos el título del programa posteriormente.

La función Dibujar_Generación_Actual()

En cada paso del bucle principal tenemos que redibujar la colonia actual de células. El pseudocódigo para hacer esto es:

```
Dibujar_Generacion_Actual:
   Para todo 'x' y todo 'y':
     Si tablero[x][y] = 0:  Dibujar Blanco en (x,y)
     Si tablero[x][y] = 1:  Dibujar Célula en (x,y)
```

Traducido a código C:

```
#define DrawCell(x,y,val) \
  *((unsigned char *) (0x4000 + 6144 + ((y)<<5) + (x))) = (val)<<3 ;

//--- Dibujar en pantalla el array de celulas ------------
void DrawLife( void )
{
   int x, y;
   for( y=0; y<ALTO; y++)
      for( x=0; x<ANCHO; x++)
      {
         Celula(x,y) = TempCelula(x,y);
         DrawCell(x,y,Celula(x,y));
      }
}
```

La clave de esta función está en la macro DrawCell, que es la que efectivamente pinta en pantalla las células. Lo interesante de la función es que no dibuja nada en pantalla, sino que modifica los atributos de la videomemoria para cambiar los "espacios en blanco" que hay en pantalla entre 2 colores diferentes (negro y azul). Concretamente, esta macro lo que hace es modificar los atributos (tinta/papel) de los caracteres de 0,0 a 32,16, accediendo directamente a videomemoria, en la zona de los atributos.

Como veremos en posteriores entregas (donde ya trataremos el tema de los gráficos), los atributos de los caracteres (tinta/papel) de la pantalla están situados en memoria a partir de la dirección 22528 (0x4000 + 6144 = 22528, es decir, tras la videomemoria gráfica de los píxeles de pantalla).

Escribiendo en esas 768 (32x24) posiciones consecutivas de memoria modificamos los atributos de los 32x24 caracteres de la pantalla. La organización es lineal, de forma que en 22528 está el atributo del carácter (0,0), en 22529 el de (1,0), en 22528+31 el de (31,0) y en 22528+32 el de (0,1), y así consecutivamente.



Simulaciones en nuestro Spectrum

Cuando escribimos un byte en una de esas posiciones estaremos modificando los atributos del caracter (x,y) de la pantalla de forma que:

Direccion_atributos(x,y) = 22528 + (y*32) + x

El byte que escribamos define los atributos con el siguiente formato:

bits 0...2: tinta (0 a 7, orden de los colores del Spectrum)

bits 3...5: papel (0 a 7, orden de los colores del Spectrum)

bit 6: brillo (1 ó 0, con o sin brillo)

bit 7: flash (1 ó 09, con o sin parpadeo)

Los colores están definidos igual que se detalla en el manual del Spectrum, es decir:

0. - negro

1. - azul

2. - rojo

3. - púrpura o magenta

4. - verde

5. - cyan

6. - amarillo

7. - blanco

Con los bits indicandos anteriormente, un atributo se construiría con el siguiente código:

```
atributo = (flash<<7) + (brillo<<6) + (papel<<3) + (tinta);
```

Por ejemplo, para establecer el caracter (2,5) con color verde (4) sobre fondo rojo (2) y con flash,

podemos utilizar el siguiente código:

```
// memaddr = 22528 + (y*32) + x
memaddr = 22528 + (5*32) + 2;

// escribir en memaddr (flash<<7)+(brillo<<6)+(papel<<3)
// +tinta.
*memaddr = (1<<7) + (2<<3) + (4);
```

De este modo podemos activar y desactivar cuadros completos de pantalla modificando su tinta y papel. Este método es mucho más rápido para nuestro programa que dibujar los 8x8 pixels de cada carácter para dibujar o apagar las células (una sóla escritura en memoria modifica el estado de 64 píxeles simultáneamente), y puede servirnos de ejemplo para mostrar cómo modificar los atributos.

La función Calcular_Siguiente_Generación()

En este momento ya tenemos una función que nos genera una colonia inicial de células (aleatoria), y un bucle principal que redibuja la colonia de células actual en memoria. Lo que falta a continuación es implementar la esencia del algoritmo de John Conway para modificar la colonia de células actual (array mapa[]) y obtener el nuevo estado (array temp[]) para, repitiendo el ciclo una y otra vez, realizar la simulación.

El pseudocódigo es el siguiente:

```
Calcular_Siguiente_Generación:
   Para todo 'x' y todo 'y':

     Si la célula es del borde
(x=0,y=0,x=ancho,y=alto):
         Matamos la célula

     Si no:
        Contar número de células vecinas.

        Si la celda (x,y) actual está
habitada:
           Si tiene menos de 2 vecinas:
              Matamos la célula
           Si tiene más de 3 vecinas :
              Matamos la célula

        Si no está habitada:
           Si tiene 2 ó 3 vecinas : Creamos
              una célula
```

El código en C que implemente este algoritmo es:

```
//--- Funcion donde se simula la vida ---------------------
void Life( void )
{
   int x, y;
   int vecinos;

   // Calculamos la siguiente generacion
   for( y=0; y<ALTO; y++)
   {
      for( x=0; x<ANCHO ; x++)
      {
        // Las celulas del borde mueren
        if( x==0 || y==0 || x>ANCHO-2 || y>ALTO-2 )
           TempCelula(x,y)=0 ;

        else
        {
           // Obtenemos el numero de celulas vecinas
           vecinos = 0;
           vecinos += Celula(x-1,y);
           vecinos += Celula(x+1,y);
           vecinos += Celula(x,y-1);
           vecinos += Celula(x,y+1);
           vecinos += Celula(x-1,y+1);
           vecinos += Celula(x-1,y-1);
           vecinos += Celula(x+1,y-1);
           vecinos += Celula(x+1,y+1);

           // reglas para células vivas
           if( Celula(x,y) == 1 )
           {
              // celulas con 2 ó 3 vecinos sobreviven
              // y el resto muere
              if( vecinos == 2 || vecinos == 3 )
                 TempCelula(x,y) = 1;
              else
                 TempCelula(x,y) = 0;
           }

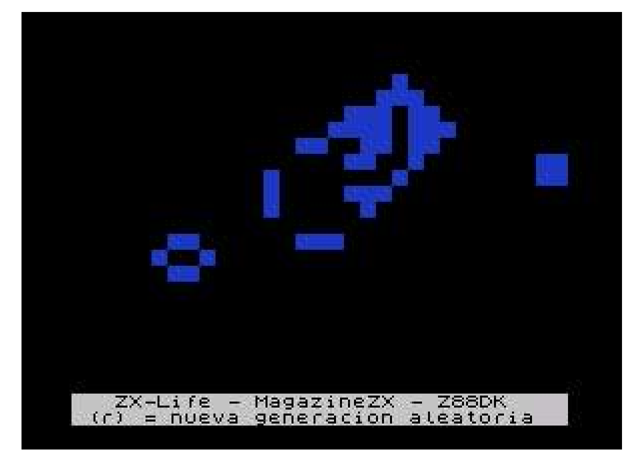
           // reglas para espacios vacios
           else
           {
              // Espacios vacios con 3 vecinos dan lugar
              // a una nueva celula
              if( vecinos == 3 )
                 TempCelula(x,y) = 1;

           } // fin else espacios vacios
        } // fin else borrar celulas del borde
      } // fin for x
   } // fin for y
}
```

Para compilar el programa puede utilizarse el siguiente comando:

```
  zcc +zxansi -vn -O1 zxlife.c -o
zxlife.bin -lndos
  bin2tap zxlife.bin zxlife.tap
  rm -f zcc_opt.def
```

Si ejecutamos el programa en nuestro Spectrum (o en un emulador) veremos la evolución de las células en tiempo real en nuestra pantalla:



Simulaciones en nuestro Spectrum

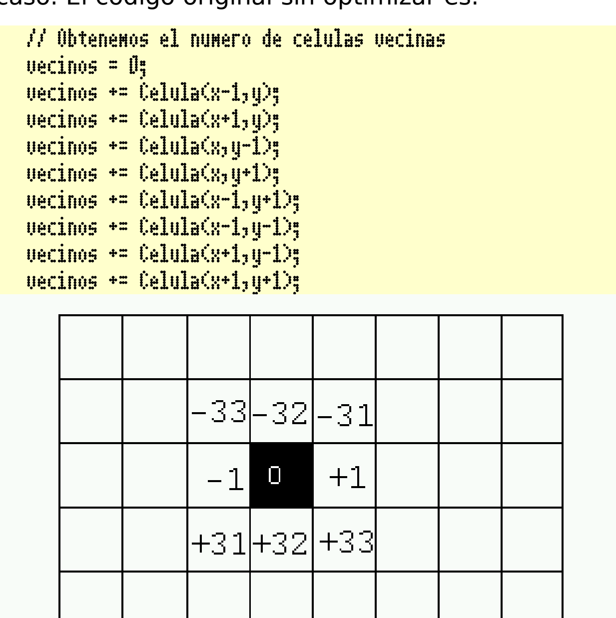
Cada vez que pulsemos 'r' se generará una nueva "remesa" de células para volver a aplicar el algoritmo y ver su evolución.

Optimizaciones del Juego de la Vida

Todos los programas pueden ser optimizados, y zxlife no es una excepción. Como ya se ha mostrado en el ejemplo del cálculo de la posición (x,y) en el array, cambiando una multiplicación por 32 por un desplazamiento binario 5 veces a la izquierda obtenemos un gran incremento de velocidad de ejecución. Este incremento de velocidad es tal porque esa función es llamada muchas veces durante la ejecución del programa. Se demuestra con esto que no por el mero hecho de utilizar C o Ensamblador el programa será más o menos rápido: La velocidad de ejecución del programa reside en que utilicemos los algoritmos adecuados a cada problema; así, desplazar binariamente 5 veces es mucho más rápido que multiplicar por 32, y esa multiplicación sería igual de lenta si la programáramos en ensamblador. De ahí la importancia del diseño del programa y los algoritmos empleados en su implementación.

En el caso de zxlife, podemos también optimizar el código de obtención de nuevas generaciones evitando cálculos innecesarios. Concretamente, cuando contamos las células vecinas podemos evitar calcular la posición de cada célula en cada caso. El código original sin optimizar es:

```
   // Obtenemos el numero de celulas vecinas
   vecinos = 0;
   vecinos += Celula(x-1,y);
   vecinos += Celula(x+1,y);
   vecinos += Celula(x,y-1);
   vecinos += Celula(x,y+1);
   vecinos += Celula(x-1,y+1);
   vecinos += Celula(x-1,y-1);
   vecinos += Celula(x+1,y-1);
   vecinos += Celula(x+1,y+1);
```



Offset de las 8 células vecinas de una dada

Cada vez que llamamos a Celula(x,y) estamos realizando el cálculo de la posición absoluta de la célula dentro del vector (en nuestra conversión bidimensional a unidimensional) mediante una serie de operaciones matemáticas. Si tenemos en cuenta que todas las células vecinas están a una distancia fija de -1, +2, -33, -32, -31 y +31, +32 y +33 de cada célula, podemos convertir esto a:

```
   offset = (y<<5)+x;

   // Obtenemos el numero de celulas vecinas
   vecinos = mapa[ offset-33 ] + mapa[ offset-32 ] +
             mapa[ offset-31 ] + mapa[ offset-1  ] +
             mapa[ offset+1  ] + mapa[ offset+31 ] +
             mapa[ offset+32 ] + mapa[ offset+33 ];
```

Esto es así porque como podemos ver en la siguiente figura, podemos obtener las 8 células vecinal a partir de un mismo offset calculado:

El código resultante de la optimización sería el siguiente:

```
//--- Funcion donde se simula la vida ---------------------
void Life( void )
{
   int x, y;
   unsigned int vecinos, offset;

   // Calculamos la siguiente generacion
   for( y=0; y<ALTO; y++)
   {
      for( x=0; x<ANCHO ; x++)
      {
        // Las celulas del borde mueren
        if( x==0 || y==0 || x>ANCHO-2 || y>ALTO-2 )
           TempCelula(x,y)=0 ;

        else
        {
           offset = (y<<5)+x;

           // Obtenemos el numero de celulas vecinas
           vecinos = mapa[ offset-33 ] + mapa[ offset-32 ] +
                     mapa[ offset-31 ] + mapa[ offset-1  ] +
                     mapa[ offset+1  ] + mapa[ offset+31 ] +
                     mapa[ offset+32 ] + mapa[ offset+33 ];

           // reglas para células vivas
           if( mapa[offset] == 1 )
           {
              // celulas con 2 ó 3 vecinos sobreviven
              // y el resto muere
              temp[offset] = 0;
              if( vecinos == 2 || vecinos == 3 )
                 temp[offset] = 1;
           }

           // reglas para espacios vacios
           else
           {
              // Espacios vacios con 3 vecinos dan lugar
              // a una nueva celula
              if( vecinos == 3 )
                 temp[ offset ] = 1;

           } // fin else espacios vacios
        } // fin else borrar celulas del borde
      } // fin for x
   } // fin for y
}
```

Aparte de la optimización de la función de cálculo de nuevas generaciones, también podemos optimizar la función que dibuja en pantalla de forma que en lugar de realizar el cálculo de posición del atributo en cada célula, lo realice una sola vez y vaya incrementándolo (algo parecido a lo

que hemos hecho con el cálculo de células vecinas). Podemos ver la versión optimizada a continuación:

```
//--- Dibujar en pantalla el array de celulas -------------
void DrawLife( void )
{
   int i;
   unsigned char *memaddr;

   memaddr = (0x4000 + 6144);
   for( i=0; i<ANCHO*ALTO; i++ )
   {
      mapa[i] = temp[i];

      // Dibujamos en pantalla cambiando el ATTR
      *memaddr = (temp[i]<<4);

      // pasamos al siguiente caracter en pantalla
      memaddr++;
   }
}
```

Lo que hacemos en el ejemplo anterior es apuntar nuestra variable puntero memaddr a la posición de memoria donde comienzan los atributos (memaddr = posición de memoria del atributo del caracter 0,0 de pantalla), tras lo cual podemos escribir el atributo e incrementar el puntero para pasar al siguiente atributo que vamos a modificar. De este modo podemos redibujar nuestros 32x16 caracteres sin tener que recalcular memaddr para cada célula, como se hacía en el caso anterior.

En el fichero comprimido que acompaña a este artículo están almacenadas las 2 versiones de zxlife: la versión 1 (zxlife.c y zxlife.tap) que es la versión original del programa, y la versión 2 (zxlife2.c y zxlife2.tap) que es la versión optimizada con los cambios que hemos explicado en esta sección.

En conclusión

Con el ejemplo de esta entrega hemos pretendido mostrar un ejemplo completo y práctico de cómo z88dk nos puede ayudar a implementar cualquier tipo de programa o algoritmo que deseemos fácilmente (zxlife.c tiene apenas 200 líneas de código contando comentarios y ha sido programado en apenas 30 minutos). Además, se ha podido ver cómo lo importante no es el lenguaje de programación utilizado, sino los algoritmos que se empleen. Por supuesto, zxlife puede optimizarse más aún: no se ha hecho porque el objetivo es que el programa fuera comprensible para los lectores, pero podemos combinar la potencia de C con funciones en ensamblador en aquellos puntos donde se considere oportuno, o utilizar una implementación diferente del algoritmo para calcular las generaciones de células, obteniendo mejores resultados.

En las próximas entregas comenzaremos a hablar de gráficos en el Spectrum, de forma que podamos comenzar a aplicar nuestros conocimientos de z88dk para hacer ya cosas visibles (gráficamente) en nuestro Spectrum.✍

LINKS

- Archivos fuente del ejemplo propuesto (zxlife.zip) http://www.speccy.org/magazinezx/revistas/10/src/zxlife.zip
- Wikipedia: http://es.wikipedia.org/wiki/Juego_de_la_vida
- Color Game Of Life Visual Exhibition: http://www.collidoscope.com/cgolve/
- What is the Game of Life?: http://www.math.com/students/wonders/life/life.html

Listado completo del programa

```
/*
 *  ZX-Life  -> Implementacion de ejemplo en C-z88dk del
 *              simulador de vida de John Conway para
 *              MagazineZX (articulo programacion z88dk).
 *
 *  v 1.0     (c) 2004 Santiago Romero AkA NoP / Compiler
 *                       sromero@gmail.com
*/
#include "stdio.h"
#include "stdlib.h"
#include "time.h"
#include "string.h"

//--- Variables y funciones utilizadas -------------------

#define ANCHO       32
#define ALTO        16
#define NUM_CELULAS 80

char mapa[ANCHO*ALTO], temp[ANCHO*ALTO];
unsigned char *memoffset;
unsigned char my_tmp_border;

#define Celula(x,y) (mapa[((y)<<5)+(x)])

#define TempCelula(x,y) (temp[((y)<<5)+(x)])

#define DrawCell(x,y,val) \
  *((unsigned char *) (0x4000 + 6144 + ((y)<<5) + (x))) =
(val)<<3 ;

void GenLife( void );
void DrawLife( void );
void Life( void );
void CLS( int value );
void BORDER( unsigned char value );


//--- Funcion principal main() --------------------------
int main( void )
{
   int i;

   GenLife();
```

```
   while(1)
   {
      DrawLife();
      Life();
      if( getk() == 'r' )
         GenLife();
   }

   return(0);
}


//--- Rellenar el tablero con valores aleat. -------------
void GenLife( void )
{
   int x, y, i;

   // Inicializar la semilla de numeros aleatorios
   srand(clock());
   BORDER(0);
   CLS(0);
   printf( "\x1B[%u;%uH",(21),(1));
   printf("   ZX-Life - MagazineZX - z88dk     ");
   printf(" (r) = nueva generacion aleatoria   ");

   // limpiamos el tablero de celulas
   for( i=0; i<ALTO*ANCHO; i++)
      mapa[i] = temp[i] = 0;

   // generamos unas cuantas celulas aleatorias
   for( i=0; i< NUM_CELULAS; i++)
   {
      x = (rand() % (ANCHO-2)) +1;
      y = (rand() % (ALTO-2)) +1;
      TempCelula(x,y) = Celula(x,y) = 1;
   }

}


//--- Funcion donde se simula la vida --------------------
void Life( void )
{
   int x, y;
   int vecinos;

   // Calculamos la siguiente generacion
   for( y=0; y<ALTO; y++)
   {
      for( x=0; x<ANCHO ; x++)
      {
        // Las celulas del borde mueren
        if( x==0 || y==0 || x>ANCHO-2 || y>ALTO-2 )
           TempCelula(x,y)=0 ;

        else
        {
           // Obtenemos el numero de celulas vecinas
           vecinos = 0;
           vecinos += Celula(x-1,y);
           vecinos += Celula(x+1,y);
           vecinos += Celula(x,y-1);
           vecinos += Celula(x,y+1);
           vecinos += Celula(x-1,y+1);
           vecinos += Celula(x-1,y-1);
           vecinos += Celula(x+1,y-1);
           vecinos += Celula(x+1,y+1);

           // reglas para células vivas
           if( Celula(x,y) == 1 )
           {
              // celulas con 2 ó 3 vecinos sobreviven
              // y el resto muere
              if( vecinos == 2 || vecinos == 3 )
                 TempCelula(x,y) = 1;
              else
                 TempCelula(x,y) = 0;
           }

           // reglas para espacios vacios
           else
           {
              // Espacios vacios con 3 vecinos dan lugar
              // a una nueva celula
              if( vecinos == 3 )
                 TempCelula(x,y) = 1;

           } // fin else espacios vacios
        } // fin else borrar celulas del borde
      } // fin for x
   } // fin for y
}


//--- Dibujar en pantalla el array de celulas ------------
void DrawLife( void )
{
   int x, y;
   for( y=0; y<ALTO; y++)
      for( x=0; x<ANCHO; x++)
      {
         Celula(x,y) = TempCelula(x,y);
         DrawCell(x,y,Celula(x,y));
      }
}


//--- Borrar la pantalla accediendo a la VRAM ------------
void CLS( int value )
{
#asm
   ld hl, 2
   add hl, sp
   ld a, (hl)
   ld hl, 16384
   ld (hl), a
   ld de, 16385
   ld bc, 6911
   ldir
#endasm
}


//--- Cambiar el borde de la pantalla --------------------
void BORDER( unsigned char value )
{
   my_tmp_border = value<<3;
#asm
   ld hl, 2
   add hl, sp
   ld a, (hl)
   ld c, 254
   out (c), a
   ld hl, 23624
   ld a, (_my_tmp_border)
   ld (hl), a
#endasm
}
```



S.T.A.R.

ALGUNOS DE MIS MEJORES AMIGOS SON GAYS, NEGROS, COMUNISTAS Y MASONES (PERO TODOS ELLOS LEEN MAGAZINE ZX)

En esta entrega volvemos a contar con la colaboración de S.T.A.R., quien creemos que no necesita presentación. Os dejamos con su artículo de opinión.

Ronald Reagan, fallecido expresidente de los EEUU, dijo una vez que un hippy es una persona que habla como Tarzán, que tiene apariencia de Jane y que huele como Chita. Nunca he acercado lo suficiente mi protuberancia nasal a un hippy como para poder aspirar su aroma pero supongo que el bueno de Ronnie debería referirse también al doble juego de palabras en el uso del verbo 'oler', de la misma forma que en nuestro país podemos decir cosas como 'este juego apesta' cuando es óbice que durante su manufacturación no ha sido tratado para provocar ningún tipo de efecto olfativo. O eso me gusta creer.

Entonces si un hippy huele como un chimpancé, que algunas visitas al zoológico su aroma real nos confirma que, efectivamente, los primates huelen a rayos, debe interpretarse (si lo tenemos en consideración) como una metáfora, refiriéndose a que el estilo y comportamiento de un hippy molesta al entorno social, y al político tambien, tal vez.

Quizás quien firma este artículo huela metafóricamente como un hippy pero no creo que su comportamiento, que no dudo que sea ofensivo y grotesco para algunos, sea tan relevante como para destacar y posicionarse por encima de la media y respetarse como una clausula generalista, la de la globalización total, de planteamientos de aldea mundial, de movimientos siempre políticamente correctos y de impersonalización estudiada, con lo que creo lícito definir al entorno como hippy más que al individuo firmante, que interpretado como definición bienquerida en aspectos e intenciones un poco más arriba apuntados, ser hippy es algo bien visto y hasta diría yo que recomendable por y para algunos. Pero son hippies, y para mí, como para Ronnie, huelen a mono.

Disentir ante cualquier opinión es un ejercicio penado, ser autor intelectual de cualquier propuesta se acompaña de un vale canjeable por una letra escarlata bordada a fuego en la frente. No seré tan simplista como para decir algo al estilo "ir en contra de", eso queda feo, amenazador y agresivo, sólo que, para que se me entienda, vivir en un chiquero el aroma es algo subjetivo en relación al lado de la valla donde uno resida. Apuesto a que Tarzán no se quejaría de los efluvios de Chita, los encontraría corrientes, habituales, en una palabra sentenciosa, normales. Pero eso no quita que el dichoso primate siguiera oliendo a rayos, de verdad.

Dentro de este ambiente de términos hasta ahora insólitos y socialmente bienaventurados por bandera, yo a mis amigos negros sigo diciendo que son negros y que los que son gays siguen siendo gays, ni los primeros son "personas de otra etnia" ni "personas de color" ni los segundos son "personas homosexuales" ni "individuos de sexualidad recíproca", ellos mismos son los primeros en ofenderse si alguien les define con verborrea y enmascara su auténtica condición. Si no fuera así, para un negro cualquier europeo sería una persona de color, y para un gay, un heterosexual sería una persona con tendencia sexual desviada. Las cosas por su nombre, que por eso gozamos de un vocabulario rico y variado.

¿Pero tan importante es la diferencia entre decir 'negro' y 'persona de color'? Pues miren, más que importante en la forma de decir me gustaría atinar más y valorar la importancia en la forma de escribir las cosas. Las palabras tienen significado, una perogrullada evidente, sí, pero a menudo es ese significado intrínseco el que se hace prevalecer sobre el orden de la sintaxis figurada, que si bien ésta se emplea instintivamente en el lenguaje hablado, encuentra réplica exacta en el lenguaje escrito cuando este pretende ser un obligado sustituto al verbo sonoro.

Textos aparecidos en esta su publicación, Magazine ZX, también han sido filibusterados por ese batido de aroma de mono y lectura distorsionada, y lo volverán a ser porque no podemos huir de esta cinta de Moebius que es tratar la escritura y su interpretación precisamente desde un medio escrito. Por cuanto uno escribe, la gente no lee, más bien traduce las letras en sonidos, los textos en palabras y se los hace llegar a su oído interno como si previamente hubieran pasado por un pedal distorsionador, haciendo como que el lector entienda lo que le viene en real gana, tanto en contexto como, atención, en tonalidad, que hay casos en los que hay para orinar y no echar gota.

El medio escrito es limitado para

reproducir expresiones. Salvo que se combinasen diferentes tipografías y se adiestre al lector a correlacionar tipos con efectos y/o entonaciones, hoy por hoy cualquier texto se tiene que interpretar, al menos partiendo con la presunción de inocencia, como un texto enunciativo. Lógico ¿verdad? Pues para un preocupante alto porcentaje de población parece que no es tan lógico. Lo que yo le diga.

Mientras se siga la linea todo es correcto, pero ay de aquel que duda, expone una alternativa que no es la linear, y no digamos ya si declara que no acepta lo hasta ahora aceptado, aún cuando lo suyo siga siendo aceptable. No es el desvío ni la independencia lo que provoca desmanes y aspavientos, no, es esa destartalada conexión que entra por los ojos, viaja a través de la red neural y llega hasta el cerebro la que interpreta como Pedro por su casa lo que el escribano había expuesto. Y cada espectador recibe una emisión diferente, el sueño de toda cadena televisiva hecho realidad, inmaterializable, por desgracia (o por fortuna) pero abstractamente sustancial, está ahí.

Si a esta percepción (o perversión) innata de las cosas le añadimos el aroma de mono, la mesa ya está servida, conseguimos una delicatessen en su punto. En el instante que en el escrito sale una palabra moral, ética o políticamente cuestionable, no se preocupen, Murphy entra en juego y el lector interpretará la peor acepción que se pueda encontrar. En el momento que en el conjunto se perciba cierta agresividad (real o ficticia, aquí el aroma de mono puede llegar a perturbar severamente) el mensaje inmediatamente será interpretado como una ofensa, una afrenta si me apuran.

¿Enfermedad del siglo XXI? ¿Efecto colateral de la sociedad de la información? Para nada, ésto existe desde siempre, sólo que ahora es cuando se populariza y se dispersa, sino, vengan, recuerden ese divertido chiste que decía algo así como:

Entra el profesor en el aula y les dice a los niños:

-Chicos, hoy va a venir un inspector del Ministerio de Educación, así que ya sabéis, portaros bien, responded a todo lo que se os pregunte y ser educados, no la arméis, que os conozco.

-Sí, señor profesor -responde toda la clase al unísono.

Al cabo del rato entra el profesor acompañado del inspector, y este se presenta:

-Hola, estoy aquí para comprobar como está el nivel escolar, y para ello llevaré a cabo un sencillo experimento que consistirá en que al primer niño, a este que está sentado delante de todo, le diré al oído una frase y él se la pasará al compañero de al lado, y este al de al lado, y así sucesivamente. Al final preguntaré salteadamente para ver lo que le ha llegado a cada niño, que ya veréis como va cambiando la cosa.

El inspector se acerca el primer niño y le dice la siguiente frase:

-Los Reyes Católicos tuvieron una hija que se llamaba Juana y que se volvió loca, por eso la llamaban Juana la Loca.

Los niños se van susurrando al oído la frase y se la van pasando hasta llegar hasta el que está sentado detrás de todo.

-Bien -dice el inspector- veamos como ha ido la cosa. A ver, tú, la niña de las coletas. ¿A tí que te ha llegado?

-Pueeeesss... a mí me ha llegado... que habían unos reyes que estaban locos y que tenían una hija que se llamaba Loca y que se estaba volviendo Juana.

-Vaya -dice el inspector- como cambia la cosa, como cambia... A ver, tú, el chaval que está sentado detrás, ¿a tí que te ha llegado?

-Sí... erg... a mí me ha llegado... erg... que había la Juana que... mmm... no estaba muy católica porque estaba volviendo locos a sus padres.

-Ya, ya veo -comenta el inspector- ¿Y a tí, al que está sentado al final de todo? ¿A tí que te ha llegado?

-No se lo digo que me pega -dice el niño que está sentado detrás de todo.

-No hombre, aquí no estoy para pegar a nadie. Dinos que te ha llegado, que a fin de cuentas es lo que te ha pasado tu compañero de al lado -le dice el inspector.

-No, no se lo digo porque me pega a mí y a mi compañero también.

-Venga, niño, no seas tonto, dinos que te ha llegado de una vez, hombre!

-Pues a mí... a mí lo que me ha llegado... a mí me ha llegado que este inspector es un desgraciado que con tantas coñas nos está jodiendo la hora del recreo.

De risa ¿verdad? Lo es porque como todo chiste juega con lo absurdo, da la vuelta a una situación que parece controlada, y sobretodo divierte porque, fíjense que ironía, se ve desde fuera, que desde dentro, tsk, tsk, no tiene ni la menor de las gracias, maldita sea.

Cuando desde estás páginas uno expone su opinión, cuando uno escribe en un medio público, ese uno cuenta con el riesgo de convertirse en ese pobre inspector del Ministerio de Educación, el desgraciado que con tantas coñas está jodiendo el recreo a los niños, y les fastidia porque están oyendo lo que no quieren oír, y ya no digamos si bajo el Minipymer metemos la interpretación distorsionada, que el efecto sería repugnantemente similar al de lanzar excrementos a un ventilador en marcha. Los hay como la niña de las coletas que interpretará el texto con neutralidad y asepticismo, no le provocará ni frío ni calor, puede darse el caso que todavía asienta y respalde al pobre inspector, pero también siempre habrá, palabrita del niño Jesús, ese o esos que le darán la vuelta a la frase o texto original y se comportarán como tiburones dando vueltas y más vueltas alrededor del tema principal, mordisco por aquí, mordisco por allá.

Esta situación me recuerda otro chiste sobre tiburones, colchonetas y bañistas despistados, pero creo que ya les he mareado bastante por hoy. No hace falta pedirles que interpreten este texto como les apetezca, eso es algo que el lector medio hace con pasmosa facilidad. ✍